Incremental Model Synchronization

by

Ali Razavi Nematollahi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2012

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Changing artifacts is intrinsic to the development and maintenance of software projects. The changes made to one artifact, however, do not come about in isolation. Software models are often vastly entangled. As such, a minuscule modification in one ripples inconsistency through several others. The primary goal of the this thesis is to investigate techniques and processes for the synchronization of artifacts in model driven development environments in which projects comprise manifold interdependent models, each being a live document that is continuously altered and evolved. The co-evolution of these artifacts demands an efficient mechanism to keep them consistent in such dynamic environments. To achieve this consistency, we intend to explore methods and algorithms for impact analysis and the propagation of modifications across heterogenous interdependent models. In particular, we consider large scale models that are generated from other models by complex artifact generators. After creation, both the generated artifacts, and also the ones they are generated from, are subject to evolutionary changes throughout which their mutual consistency should be maintained. In such situations, the model transformation is the primary benchmark of consistency rules between source and target models. But the rules are often implanted inside the implementation of artifact generators and hence unavailable. Trivially, the artifacts can be synchronized by regeneration. More often than not however, regeneration of such artifacts from scratch tends to be unwieldy due to their massive size. This thesis is a summary of research on effective change management methodologies in the context of model driven development. In particular, it presents two methods of incrementally synchronizing software models related by existing model transformations, so that the synchronization time is proportional to the magnitude of change and not to the size of models. The first approach treats model transformations as black-boxes and adds to it incremental synchronization by a technique called conceptualization. The black-box is distinguished from other undertakings in that it does not require the extraction, re-engineering and re-implementation of consistency rules embedded inside transformations. The second approach is a white-box approach that uses static analysis to automatically transform the source code of the transformation into an incremental one. In particular it uses partial evaluation to derive a specialized, incremental transformation from the existing one. These two approaches are complementary and together support a comprehensive range of model transformations.

Acknowledgements

I would like to acknowledge my profound, but hopefully not prolix, gratitude to many individuals whose support and advice were greatly conducive to this project.

Kostas Kontogiannis, my PhD supervisor, has provided me with his gracious support for over eight years, not just for overcoming academic difficulties, but in matters that shape one's life and character. I truly feel indebted to him. Ladan Tahvildari, my PhD co-supervisor, has helped me on numerous occasions academically and otherwise by graciously sharing her invaluable personal experiences with me. I would like to express my sincere gratitude to professors Rudolph Seviora, Paulo Alencar and Derek Rayside for agreeing to serve on my PhD committee and for taking the time to read my proposal and this dissertation. Special thanks goes to professor Eleni Stroulia from the university of Alberta, for agreeing to be my external examiner.

Funding for this research was in part provided by IBM Center for Advanced Studies. As a IBM CAS PhD Fellow, I had the privilege to attend three summers in IBM Canada Lab which was a fulfilling experience. I would like to thank Chris Brealey for seeding the initial idea of the project and providing insightful feedback and valuable collaboration along the way.

I would like to express my sincere gratitude to Professor Ali Safavi, whom I have been fortunate enough to have as a roll model since my early childhood. I very much appreciate the father-like advices and emotional supports he has given me time and again throughout these years. Finally, nothing can *synchronize* a prose attempting to describe my love and admiration for my parents with the real sentiment, depth of which goes far beyond the expressive power of words.

Dedication

Dedicated to my parents.

Contents

List of Figures **xiii**

Glossary **xiv**

1 Introduction **1**

1.1 Problem Description 3

1.2 Summary of Contributions 6

1.3 Thesis Outline 7

2 Related Work **9**

2.1 Perspective on Model Change 9

2.2 Change Propagation 9

2.3 Metamodel Evolution 10

2.3.1 Model Transformations 10

2.3.2 Model Consistency and Dependency 13

2.4 Model Refactoring 15

2.5 Partial Evaluation 15

3 Characterization of Change **17**

3.1 Model Elements, Models and Metamodels 17

3.1.1 Equivalence of two model elements 22
3.2 Atomic Model Manipulation Operators 25
3.3 Change Factorization and Model Edit Distance 27
3.3.1 Edit Script Normalization 33
3.3.2 Impact set of a Change 34
3.3.3 Model Transformations 41
3.3.4 Classification of Transformations 41
3.3.5 Traces and Dependencies across Transformations 47
3.3.6 Semantics of Synchronization for Generated Artifacts 48
3.3.7 Bi-directional synchronization 48
3.3.8 Incremental synchronization 49

4 Incremental Synchronization of Black-box Transformations 50
4.1 Architecture Overview 51
4.1.1 Overall Process 53
4.1.2 Conceptualization Phase 56
4.1.3 Shadow Phase 57
4.1.4 De-Shadow 61
4.1.5 Synchronization Process 62
4.2 Insertion and Deletion 67
4.2.1 Propagation of Insertion Induced Changes 67
4.2.2 Propagation of Deletion 76
4.2.3 Dependency Inference 76
4.3 Complete Picture for The Running Example 78
4.4 Properties of the Black-box Synchronizer 81
4.4.1 Termination 81

4.4.2 Transformation Chains 82
4.4.3 Soundness 86
4.5 Complexity 87
4.6 Experiments and Evaluation 88
4.6.1 Experiment Setup 88
4.6.2 Results 89
4.7 Application and Integration 91

5 White-Box Incrementalization of Transformations 96
5.1 Introduction to Partial Evaluation 97
5.2 Introduction to Query View Transformation Operational Mappings 101
5.2.1 Model Transformation with QVT Operational Mappings 102
5.3 Synatx of Essential QVT-OM 106
5.3.1 Abstract Syntax 106
5.4 Overall Process and System Architecture 107
5.4.1 Partial Evaluation of Model Transformations 111
5.5 Partial Evaluation 114
5.5.1 Offline Binding Time Analysis Rules 115
5.5.2 Online Binding Time Analysis Rules 118
5.5.3 Constant Propagation and Static Expression Evaluation 119
5.5.4 Loops 119
5.5.5 Conditional Expressions 121
5.5.6 Symbolic Expression Simplification 122
5.5.7 Reduce Rules 123
5.5.8 Contexts and Function Calls 124
5.5.9 Reducing Iterate Expressions 127

5.5.10 Mixed Reduction of Partially Static Collections 128
5.5.11 Mix Rules 131
5.5.12 Partial Evaluation of Mapping Operations 135
5.5.13 Polyvariant Mix Rules 140
5.5.14 Polyvariant Reduction of Iterate Expressions 141
5.6 A Full Example 143
5.7 Experiments and Discussion 147

6 Conclusion **151**
6.1 Future Work 152
6.1.1 Information Content of Models and Model Transformations 152
6.1.2 Improved Conceptualization Schemes 153
6.1.3 Automatic Generation of Abstractors/DeAbstractors 153
6.1.4 Embedded Rule Language for Composite Concepts 154
6.1.5 Fine-Grained Version Management Using Concepts 154
6.1.6 Enhancements to QvtMix 154
6.1.7 Self Applicability of QvtMix 155

Appendix **156**

A Haskell Implementation of Change and Sync **156**

B Abstract Syntax of QVT Operational Mappings **170**

C Big-Step Operational Semantics of QVT Operational Mappings **174**
C.1 Big-Step Operational Semantics 174

D QvtMix Implementation **182**

E Change Factorization Algorithm Implementation in C++ **303**

Bibliography **317**

List of Figures

1.1 Problem Description 4

3.1 Sample Model Abstracting Java Code 18

3.2 Runtime and Space Complexity of Dynamic Programming Change Factorization Algorithm. X-Axis is n, the size of the randomly generated model, and Y-Axis measures the value of $O(d)$ as represented in Theorem 3.3.1 with a constant of 3, the actual size of the table is represented by $|d|$, the product of the size of two models is represented by $|s|.|t|$ for comparison. T is the runtime of the algorithm which is normalized and projected for comparison with the space complexity. 33

4.1 Architecture of the Synchronization Framework (arrows denote dataflow) 52

4.2 Generating WSDL from Java Classes 53

4.3 Simplified metamodel for Java Implementation of a Web Service 53

4.4 Simplified metamodel of WSDL 2.0 54

4.5 Abstract Models of Java and WSDL 55

4.6 Conceptualization of Java and WSDL models 58

4.7 Creating Shadow for Java model 60

4.8 Transformation of Java Shadow to obtain WSDL Shadow 61

4.9 Synchronization by Shadow 63

4.10 Injecting μ-templates 70

4.11 Transformation of μ-templates 71
4.12 μ-template Consumption 73
4.13 Propagating Insertion 75
4.14 Black-Box Synchronization Process 79
4.15 Spiral Synchronization of Dependency Cycles 83
4.16 Mutli-step consistency establishment 84
4.17 Performance Evaluation(Log Y-Axis) 90
4.18 Performance for multiple beans 91
4.19 Shadow files space overhead 92

5.1 Partial Evaluation (adapted from [42]) 99
5.2 Query View Transformation stack of model transformation languages . . . 101
5.3 Sample source metamodel MM and its instance M 103
5.4 Transformation `T` mapping `M:MM` to an instance of the target metamodel `NN` 103
5.5 System Architecture Diagram 109
5.6 Partial Evaluation of Model Transformations 112
5.7 Monovariant Mix Reduction 129
5.8 `PUB` and `BOOK` Metamodels 136
5.9 Annotations for static analysis 137
5.10 The Source and Target Instance Models 144
5.11 Applying the Specialized Transformation to the Changed Source Model . . 147
5.12 Execution time of the original and specialized transformation for growing input sizes 149
5.13 Execution time of the original and specialized transformation based on the utilization of input elements for partial evaluation 150

B.1 Simplified Metamodel of QVT Operational Mappings 171

B.2 Simplified Metamodel of Imperative OCL 172
B.3 Simplified Metamodel of OCL 173

Glossary

$\therefore$ therefore.

$(a_1, .., a_n)$ n-tuple consisting of elements $a_1, .., a_n$.

$\langle e_1, .., e_n \rangle$ list (ordered sequence) of elements $e_1, .., e_n$.

$|A|$ size of a set or list.

$\pi_i(S)$ projection of the i_{th} element of a tuple or list.

$\{m_1, .., m_n\}$ set of elements $m_1, .., m_n$.

$\varnothing$ empty set or list.

$a \in S$ set or list membership.

2^A power-set (set of all subsets) of A.

$A \times B$ cartesian product of sets A and B.

$A \cup B$ union of sets A and B.

$S \uplus T$ catenation of lists S and T.

$f : A \to B$ function f with domain A and co-domain B.

$f \circ g$ function composition $f \circ g(x) = f(g(x))$.

$f \,;\, g$ function composition $(f \,;\, g)(x) = g(f(x))$.

$\perp$ undefined value.

$\square$ place holder for any value.

$O(\mathfrak{g}(\mathfrak{n}))$ big-O, asymptotic upper bound growth of $\mathfrak{g}(\mathfrak{n})$.

$\mathbb{N}$ Natural numbers (positive integers).

Σ Alphabet of symbols.

Σ^* Transitive closure of alphabet Σ. Set of all possible permutations.

$\mathcal{L}$ language $\mathcal{L} \subseteq \Sigma^*$ (c.f. Σ^*).

$\mathcal{L}(\mathsf{MM})$ language generated by meta-model MM, set of all instances.

$(\mathtt{Sig}_\mathsf{C}, \mathtt{Sig}_\mathsf{A}, \mathtt{Sig}_\mathsf{R})$ metamodel with signature functions for containers, attributes and references, respectively.

$(\mathfrak{C}, \mathsf{A}, \mathsf{R}, \mathsf{T})$ model element of type T with container list $\mathfrak{C}$, attribute list A and reference list R.

$\nu : \mathsf{T}$ value ν has type T.

$\mathcal{T}$ set of all types.

$\mathcal{M}$ set of all models.

$\mathcal{C}$ set of all containers.

$\mathfrak{C}$ list of containers of a model element (c.f. $\mathsf{C}_\mathtt{i}$).

$\mathsf{C}_\mathtt{i}$ the $\mathtt{i}_{\mathtt{th}}$ container.

$\mathfrak{m}/\mathtt{i}$ the $\mathtt{i}_\mathtt{t}\mathtt{h}$ container of model $\mathfrak{m}$.

$\mathfrak{c}.\mathtt{j}$ the $\mathtt{j}_{\mathtt{th}}$ element of container $\mathfrak{c}$.

$\mathfrak{m}@\mathtt{k}$ the $\mathtt{k}_{\mathtt{th}}$ attribute of model $\mathfrak{m}$ (or attribute named $\mathtt{k}$).

$\mathfrak{C}(\mathfrak{m}), \mathsf{A}(\mathfrak{m}), \mathsf{R}(\mathfrak{m}), \mathsf{T}(\mathfrak{m})$ containers, attributes, references and type of model $\mathfrak{m}$, respectively.

$\mathtt{Addr}_\mathsf{M}(\mathfrak{m})$ Address of element $\mathfrak{m}$ in model M.

$\cup\,\mathfrak{C}$ union of all containers in $\mathfrak{C}$.

$\uplus\mathfrak{m}$ all elements contained by $\mathfrak{m}$.

$\mathfrak{m} = \mathfrak{n}$ equivalence of models $\mathfrak{m}$ and $\mathfrak{n}$.

$\mathfrak{m} \overset{\mathsf{M},\mathsf{N}}{\equiv\!\equiv} \mathfrak{n}$ congruence of model elements $\mathfrak{m}$ and $\mathfrak{n}$ residing respectively in models M and N.

$\mathfrak{m} \cong \mathfrak{n}$ weak equivalence of $\mathfrak{m}$ and $\mathfrak{n}$.

$\blacklozenge(\mathfrak{m}, \mathfrak{i}, \nu)$ update the value of the $\mathfrak{i}_{\mathfrak{th}}$ attribute of model element m to ν.

$\blacklozenge(\mathfrak{m}@\mathfrak{a} \mapsto \nu)$ update the value of attribute $\mathfrak{a}$ of model element $\mathfrak{m}$ to ν.

$\blacklozenge^{\perp}$ Reset operator, changing all values of its argument model to $\perp$.

$\blacktriangle(\mathfrak{m}, \mathfrak{i}, \mathsf{T})$ insert new element of type T into the $\mathfrak{i}_{\mathfrak{th}}$ container of model element $\mathfrak{m}$.

$\blacktriangle(\mathsf{M}/\alpha)$ insert into container address M/α.

$\blacktriangledown(\mathfrak{m}, \mathfrak{i})$ delete the last element of the $\mathfrak{i}_{\mathfrak{th}}$ container of model element $\mathfrak{m}$.

$\blacktriangledown(\mathsf{M}/\alpha)$ delete the last element of the container at address α.

$\mathcal{I}^{\Delta}_{\mathsf{M}}$ impact of change Δ on model M.

$\mathfrak{m} \overset{\delta}{\rightsquigarrow} \mathfrak{n}$ $\mathfrak{m}$ morphed into $\mathfrak{n}$ by change δ.

$\delta \xrightarrow{\mathsf{T}} \delta'$ change δ translated to δ' by transformation T.

$\mathtt{Clone}(\mathsf{M})$ duplicate model M (and all its children).

$\mathcal{CP}$ concept pool.

$\mathfrak{a} \hookrightarrow \mathfrak{c}$ attribute $\mathfrak{a}$ points to concept $\mathfrak{c}$.

μ micro template.

$[\![\mathtt{prog}]\!]_{\mathcal{L}}[\mathtt{in}] = \mathtt{out}$ program $\mathtt{prog}$, interpreted in language $\mathcal{L}$, produces $\mathtt{out}$ from $\mathtt{in}$.

$[\![\mathrm{x}]\!]$ x quoted as expression.

Exp set of all expressions.

Env set of all environments.

τ binding-time environment.

α annotation environment.

Γ type environment.

σ variable store.

$\sigma[a \mapsto v]$ new store mapping a to v and every other x to $\sigma(x)$.

S static binding-time.

D dynamic binding-time.

M *maybe* static binding-time.

VAR variable annotation.

$\mathsf{VAR}^{\blacktriangle}$ insertion change annotation.

$\mathsf{VAR}^{\blacktriangledown}$ deletion change annotation.

$\mathsf{VAR}^{\blacklozenge}$ update change annotation.

FIXED Invariant annotation.

$t_1 \sqcap t_2$ binding-time conjunction.

$t_1 \sqcup t_2$ binding-time disjunction.

$\mathcal{B}[\![.]\!]$ online BTA function.

$\mathcal{E}[\![.]\!]$ eval function.

$\mathcal{SE}[\![]\!]$ side-effect eval.

$\mathcal{R}[\![.]\!]$ reduction.

$\mathcal{M}[\![.]\!]$ mix reduction.

$\mathcal{PM}[\![.]\!]$ poly-variant mix reduction.

$e \curlyvee e'$ merge of expressions e and e'.

$\mathtt{Context}$ set of all contexts.

ψ memoization table.

Ψ space of memoization tables.

ζ helper environment, keeping the description of all helper functions.

ξ mapping environment, keeping the description of all mapping functions.

$\sigma; e \Downarrow \nu$ large-step reduction; expression e reduces to value ν in environment σ, which will remain unchanged.

$\langle \sigma, e \rangle \Downarrow \langle \sigma', \nu \rangle$ large-step reduction; expression e reduces to value ν in environment σ, during which it will be mapped to σ'.

$\_\_$ *don't care.*

$\circlearrowleft$ looping flag.

$\lightning$ break flag.

$\downarrow$ proceed flag.

$\curvearrowleft$ return flag.

Chapter 1

Introduction

The principal aspiration of Model Driven Development (MDD) is to raise the level of abstraction in software development. Models are denoted using diagrams and graphical notations. As such, they potentially are able to convey requirement and design information more succinctly in comparison with textual formats. MDD aims to let the stakeholders of all levels concentrate on solving the problem at hand, rather than overcoming the difficulties imposed by low-level intricacies of computation [44, 9, 68, 55]. Compared with code-centric approaches, the model driven paradigm of software development is shown to significantly boost productivity in spite of its relative immaturity (for a comprehensive empirical evaluation of UML effectiveness see [23]). In its envisioned usage, MDD deems requirement, architecture and design documents as live and evolving artifacts [38]. This comes in contrast with the traditional software development lifecycles during which such models are often inanimate documents.

Model driven projects, compared with code centric ones, encompass more diverse types of artifacts. The heterogeneity of MDD environments stems from several key characteristics of this approach towards software development. First, modeling facilities, such as UML, offer an assortment of means for specifying orthogonal aspects of a system. UML version 2.0, for example, provides 13 different types of diagrams [56], each dedicated to model a single facet of the system. In particular, UML Class diagrams describe the structural decomposition of classes and their static relationships with each other. Other examples are Sequence diagrams whose primary intent is to model dynamic interactions between objects, and Deployment diagrams that describe the logistics and distribution of software

components. These diagrams share both latent and obvious cross dependencies. For example, the objects modeled in a sequence diagram have the signature of their classes defined in a corresponding class diagram. Therefore, altering the name of a class, for example, requires the sequence diagram and other related models to be updated accordingly.

Second, MDD advocates the encapsulation of domain knowledge in the form of domain specific languages(DSL). The syntax and semantics of these languages are usually specified as models themselves which in part are assets to projects that take advantage of them. In addition, there are domain specific models specified in these languages. Because the DSLs being used for modeling are often developed in-house, they are prone to change frequently. Changing a language compels its instances to comply with its new syntax or semantics, thus they should also be modified accordingly.

The third reason for the diversity of artifacts is the support for multiple platforms needed in many recent projects. Software systems are becoming more distributed which means there are more than one architecture involved in most projects. Usually a variety of platforms with different operating systems and middle-wares are deployed. Each platform, to play its particular role in the operation of the system, depends on various platform specific documents, such as deployment descriptors, schemas etc., for proper configuration and deployment. MDD, because of its promises to abstract the complexity of various platforms, is often touted as the paradigm of choice for developing such projects. Consequently, the platform specific documents are also treated as models, hence making the projects even more multifarious.

Throughout their life-cycle, software artifacts are iteratively changed. Being logical interdependent entities, changes made to one model would impact other models. Modern development environments strive to provide facilities for transparent and effective propagation of changes across the workspace. Being diverse however, means that the task of synchronizing models is significantly more challenging in MDD environments in which manifold of models are subject to miscellaneous transformations. To ensure the *homeostasis*[1] of systems, their models ought to remain consistent according to an assortment of rules imposed by languages, grammars, meta-models, constraints and transformations.

A development environment should be responsive and interactive. They are meant to help the developers achieve their goals without impeding their productivity. Change

[1]Software Homeostasis in the sense that Mary Shaw in [70] has defined as: *"Homeostasis is the mechanism through which a system acts to maintain a stable internal environment despite external variations."*

propagation and subsequent model synchronization tasks thus have to be carried out as transparently to the user as possible. These requirements motivate that the synchronization of large models be performed incrementally, i.e., the synchronizer should only reconcile the affected fractions of models and bypass the unchanged elements. The system should also facilitate analyzing the impact of a change, and let developers preview its alternative outcomes before conducting it. This analysis is particularly useful for manipulating large models.

Generative software development uses transformations to create software artifacts of different types from one another [21]. The consistency rules of these artifacts are often embedded inside the source code of the artifact generator. These generators are not always thoroughly documented. Moreover, development processes occasionally integrate third party transformations , no information about whose internal logic is available. For the same reasons mentioned earlier, to synchronize efficiently the models generated with these model transformations, is a desirable feature. Nevertheless, reverse engineering the transformation rules for these artifact generators tend to be project specific and take time and resources. Any update propagation method capable of incrementally reconciling generated models–without the need to reverse engineer and re-implement them–has considerable utility.

1.1 Problem Description

The objective of change management is to co-evolve a set of models whose consistency is disturbed by local changes towards a new consistent state. Consistency between two interdependent models can, in general, be described using a relationship defined over the state space of the mutations of those models. Change management process relies on the ability to synchronize two models based on a given consistency relationship between them.

Change management relies on synchronization as one of its fundamental building block. Therefore, to tractably manage changes between large models with complex relationships, model synchronization has to be incremental. This, in principle, requires the complexity of propagating changes be proportional to the size of changes, and not that of the source and target models. In other words, the synchronizer should manipulate only the parts that are affected by the changes; any redundant rewrite of the parts that need no modifications

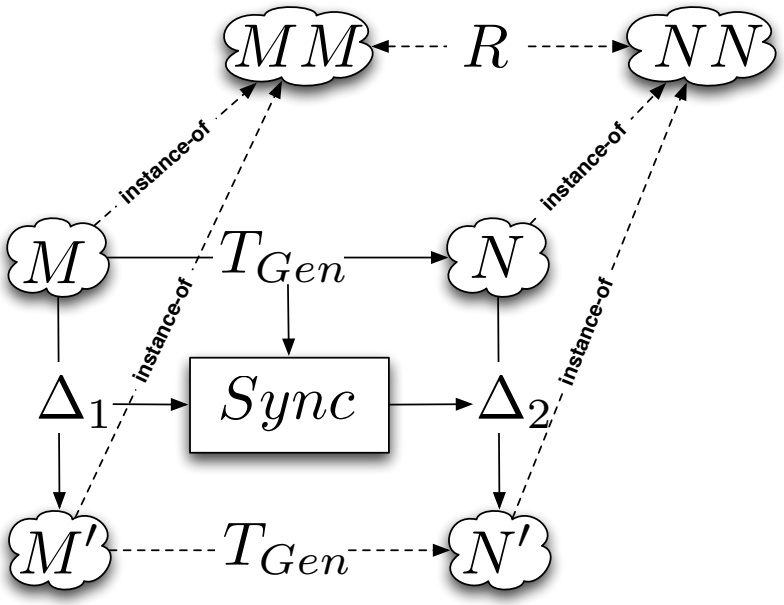


Figure 1.1: Problem Description

should be avoided. This, to some extent, is analogous to the concept of compilation with the *make* utility, which only recompiles the most recently changed modules. Only, the level of granularity for model synchronization is reduced down to the individual model elements from code modules in the case of program compilation.

Figure 1.1 illustrates the problem of model synchronization in a more precise manner. In this N is the target model which is generated by applying transformation $\mathsf{T}_{\mathtt{Gen}}$ on the source model, namely M. The consistency relationships between models can originate from different sources. One particularly important category is when these relationships are implicitly enforced by a model transformation. For example in Figure 1.1, the consistency requirement is connoted by a transformation that has originally been used to generate (or update) the target model from the source model. As such, future consistency, when needed, can be established by re-invoking that transformation. Although, these transformations are often non-incremental– hence suffer from the problems mentioned earlier–they, nonetheless, can serve as an objective measure of correctness for change propagation. This, in other words, asserts that model synchronizers should incrementally produce updated models that are compatible with the result of the original model transformation, should it run again

over the altered source model.

Transformations can be mathematically described as mappings between domains, which are specified by metamodels. Analogous to the terminology of formal languages, a metamodel (or a domain model) can be considered as a generative grammar for a domain; $\mathcal{L}(\mathsf{MM})$ denotes the language generated by metamodel specification MM, i.e., the (possibly infinite) set of all models conforming to MM.

Changes in artifacts can be described as endogenous transformations, that is, transformations whose source and target models conform to the same metamodel. $\Delta : \mathcal{L}(\mathsf{MM}) \mapsto \mathcal{L}(\mathsf{MM})$ defines transformation Δ applicable to models conforming to metamodel MM. Given this, interdependent model synchronization becomes the problem of finding the most concise change operation $\mathsf{N}' = \Delta_2(\mathsf{N})$ applicable to model $\mathsf{N} \in \mathcal{L}(\mathsf{NN})$ for change $\Delta_1(\mathsf{M}) = \mathsf{M}'$ applied on model $\mathsf{M} \in \mathcal{L}(\mathsf{MM})$, such that N' and M' remain consistent according to consistency criteria the same as those between M and N. By concise, we mean that Δ_2 should consist of the minimal number of change operations whose collective impact is confined to the affected elements of N.

Artifact generation is an example of exogenous model transformation, that is, its source and target models adhere to different metamodels. Mapping $\mathsf{T}_{\mathtt{Gen}} : \mathsf{MM} \mapsto \mathsf{NN}$ transforms source models that conform to metamodel MM to target models conforming to metamodel NN. Assuming that models M and N are synchronized–i.e., they are consistent according to some relation R–a simple approach to re-synchronizing M' and N is to re-apply $\mathsf{T}_{\mathtt{Gen}}$ on M' to obtain N'. However, when M is large enough relative to Δ_1, and/or $\mathsf{T}_{\mathtt{Gen}}$ is computationally expensive, this approach based on indiscriminate re-generation would not be sufficiently performant for modern, interactive development environments. Furthermore, regeneration does not provide any means for reflecting the changes made to the target model back to the source model. As the diagram depicts, a mapping such as Sync could be used to generate from Δ_1 a sequence of change operations, Δ_2, which is applicable to N, and yields a model identical to $\mathsf{N}' = \mathsf{T}_{\mathtt{Gen}}(\mathsf{M}')$, which is consistent with M' according to the same relation, R. We say that Sync is incremental, if the number of elements in N that Δ_2 modifies to obtain N' is minimal. Thus, the problem of model synchronization for the models bound together with transformation rules such as $\mathsf{M} = \mathsf{T}_{\mathtt{Gen}}(\mathsf{N})$ reduces to determining $\Delta_2 : \mathsf{MM} \mapsto \mathsf{NN}$ for every change Δ_1 defined on the source model, such that $\Delta_2(\mathsf{N}) = \mathsf{T}_{\mathtt{Gen}}(\Delta_1(\mathsf{M}))$.

1.2 Summary of Contributions

In this section we present an outline of the major contributions made in this research work[2].

- **Foundations**
  - Abstract and formal characterization of models, metamodels, transformations and change operations. We define a succinct notation to express MOF like models using mathematical constructs. We also define atomic change operations, which serve as building blocks for composing general manipulation of model structures.
  - Two efficient algorithms for computing edit distance of models based on the insert/delete and the append/drop change operations. The first algorithm leverages the definitions of models and atomic change operations to first find a series of changes to make both models structurally similar, and then update attribute values where there are descrepencies. The second algorithm uses a dynamic programming approach to find a minimum-cost edit script based on the cost associated with each atomic change operation.
  - Algebraic solution for the simplification of composite change scripts. This methodology is used to obtain canonical forms for edit scripts, in which changes whose effects are cancelled by later changes in the script are eliminated.
  - Characterization of the impact set of a change and presenting an algebraic, mechanical solution for change impact analysis, whereby making it possible to calculate the impact-set of a complex change *a priori* before applying the change operations.
  - Classification of model transformations based on their change translation behavior and proving several important properties
  - Formal definition for incremental and bi-directional synchronization schemes
  - Canonical implementation of models, metamodels and change operations in Haskell

[2]Parts of this work has been published in these peer-reviewed conferences [61, 63, 62], and, at the time of this writing, several hitherto unpublished parts are being prepared as manuscripts for submission.

- **Black-box synchronization of model transformations**
  - Sync: a novel methodology for the synchronization of blackbox model transformations
  - Analysis of completeness, soundness and efficiency of the proposed black-box synchronization scheme
  - Canonical implementation of black-box synchronization in Haskell
  - Implementation and integration with Eclipse Web Tools Platform (WTP) and application to real-world transformation scenarios of service oriented artifacts
  - Conducting experiments for performance evaluation of the framework
- **White-box synchronization of model transformations**
  - Methodology for the White-box synchronization of imperative model transformations based on partial evaluation
  - Design and specification of several reduction algorithms for the specialization of model transformations
  - Design and implementation of QvtMix: a hybrid partial evaluator for QVT Operational Mappings implemented in QVT itself
  - Big-step operational semantics for a subset of QVT Operational Mappings
  - Experiments for the characterization of the performance and the space overhead of the partial evaluator

1.3 Thesis Outline

The remainder of this dissertation is organized as follows. A comprehensive survey of related research and technologies is presented in the next chapter. Chapter 3 presents preliminary materials and establishes the foundation for the rest of the thesis. It includes, among other topics, the discussion of change factorization algorithms, change script simplification, change impact analysis and the definition of incrementality for model synchronization. Chapter 4 introduces the black-box synchronization techniques, and discusses various properties of the black-box framework. Chapter 5 gives an elaborate account of

the proposed techniques for white-box model synchronization. QvtMix, a partial evaluator for the QVT Operational Mappings transformation language, is presented to introduce several partial evaluation techniques, their application in the context of model transformations, and how they aid us achieve incrementality for model synchronization scenarios. Chapter 6 summarizes the dissertation and confers on several avenues for further research.

We believe that, in the age of inexpensive bits and storage, no software engineering manuscript is complete without supplementary proof of concept implementation code. This helps make the document self-contained, and improves the reproducibility of claimed results. As such, we present canonical implementation for the crux of the algorithms presented here in three appendices. More specifically, the minimalist and semi-formal notation we have established for models and change operators, along with the most fundamental parts of the black-box synchronization framework, are implemented in Haskell, which is presented in Appendix A. Appendix D presents the elaborate implementation of QvtMix. The code for the dynamic-programming change factorization algorithm is presented in Appendix E. Moreover, throughout the main body of the thesis, various pointers have been inserted to the electronic version of this document, so as to enable the reader quickly navigate to pertinent parts of the implementation.

Some peripheral material regarding the QVT-OM language are provided as appendix: an elaborate abstract syntax of the language which illustrates its relationship with other parts of OMG's ecosystem of modeling standards (in particular, MOF and OCL) are presented in Appendix B. Appendix C specifies the semantics of the subset of the QVT-OM language used for discussing the design of QvtMix.

Chapter 2

Related Work

2.1 Perspective on Model Change

There are two major perspectives on change. The first one, the *state-based* view, considers changes as morphisms that map the state of the model they are applied upon to a new state in the general state-space of instance models outlined by the metamodel. The second school of thought is what is referred to as the *operational* view, which defines a set of change operations and represents the differences between models as a sequence of those operations. Examples of frameworks that have adopted the latter representation include: Lenses in Harmony [28], Rondo [50], and most graph transformation based frameworks, e.g., AGG and VIATRA. Alternatively, some examples of frameworks which incorporated the operational perspective include: the works of Porres and Alanen in [60] and [4], ATL and SyncATL [81], Deltaware [36], Beanbag [82] and the paper by Blanc et al [13]. Our notion of change, introduced in Chapter 3, is operational, which defines 5 primary atomic change operations composeable into arbitrary changes.

2.2 Change Propagation

Model driven projects typically comprise several inter-related models, and therefore modification of a model may cause several violations of the consistency relationships between the inter-related models in a project. To reconcile the system back into a consistent state,

it is needed to appropriately *propagate* the changes to other models that are invalidated by the recently performed change operations. Other terms commonly used for change propagation in the MDD community are: *Model Synchronization* [31], [81] and [6], update transformation [60], update propagation [17] and update translation [28].

2.3 Metamodel Evolution

In MDD, everything is a model. This general rule also applies to metamodels and, therefore, they can also be modified using the same scheme developed for changing models. However, metamodels are essentially grammars that describe a space of instances. As such, changing a metamodel can invalidate some of its former instances, extend the instance space of said metamodel, or do a combination of both. Changing metamodels along with the repercussions of such changes on their instances have been explored in the literature. Wachsmuth proposes definitions for *metamodel adaptation* and *model co-adaptation* in [78], in which some important properties of metamodel changes, such as *Instance Preservation*, are discussed. Another related term in this context is *Model Co-evolution*, which is used to describe the changes of instance models along with those of their metamodels in a coupled evolutionary development process, during which the conformance of models to their metamodels have to be preserved [16], [37] and [45]. In this paper, we have assumed that during the life-cycle of the software artifact maintained by the synchronization framework, their metamodel remains unchanged. However, supporting such changes, is a feature worthy of future research.

2.3.1 Model Transformations

The major premise of our work is to avoid re-implementation of an existing transformation in another language or framework. This sets an immediate diverging point between our approach and that of other incremental synchronization frameworks, majority of which define some sort of specification mechanism, whereby an existing transformation has to be re-implemented. In contrast, we take the transformation's implementation as a black-box, and build the support for incremental synchronization around it, only assuming a few generic properties about the transformation.

A catalogue of various model synchronization schemes is offered in [6]. The paper focuses on formulating the external behavior of model transformations. A general classification of model transformation frameworks can be found in [20]. Two features that specifically concern the problem of model synchronization are bi-directionality and incrementality. The paper introduces several frameworks that support either or both of these features, most notable of which is the Object Management Group's Query View and Transformation (QVT) [10]. QVT proposes the *Relation* and the *Core* languages, both of which capable of denoting incremental and bi-direction transformations with different expressiveness and complementary levels of abstraction.

Triple Graph Grammar (TGG) is a graphical, declarative, incremental and bi-directional model transformation methodology based on graph transformation [67]. It has been advocated to be an effective foundation for tool integration [5]. Beanbag is an emerging framework which offers a language that supports intra-relations between models [82]. Our conceptualization scheme also supports intra-relations.

Bi-directional transformations and their application in model synchronization have been investigated by researchers in the programming languages community. Foster et al. proposed the Harmony framework based on the notion of *Lenses* for bi-directional synchronization [28]. They propose a language in which programs are inherently bi-directional. The Harmony framework has a state-based perspective on change. Contrariwise, Alanen and Porres have investigated syntactic merging, differentiation and union of structural models in [4] by adopting an operational and compositional view of changes.

One solution that works with existing transformations is SyncATL, proposed by Xiong et al. in [81]. They have extended the bytecode of the ATL virtual machine [8], whereby supporting automated backward synchronization of models linked by an ATL transformation. For the forward synchronization, SyncATL relies on re-invoking the transformation and merging the results with the existing target. The proposed technique, however, does not address incremental synchronization at all, as the framework relies on re-executing the transformation for forward change propagation. The other drawback of this approach is its tight integration with a specific technology, i.e., ATL.

Another framework with the theme of building incremental synchronization around existing transformation engines is presented in [36]. This time the Tefkat transformation engine, which has logical programming flavor, is decorated with support for incrementality.

In Tefkat, transformation rules are specified as logical predicates, and are performed using SLD resolution. Their synchronization framework avoids redundant computation in successive transformation of the same model by preserving the intermediate SLD trees. Another logic based framework which exploits answer set programming for change propagation is presented in [15].

In [76], Tratt articulates the spectra of challenges involved in model driven tool integration with model transformations being their centerpiece. The paper stresses the importance of the particularly challenging problem of change propagation and enumerates the reasons for inadequacy of solutions based unilaterally on commonly championed panaceas such as bidirectional transformations or traceability. He concludes that a comprehensive change propagation scheme, to be pragmatically effective, should function tractably over the most common patterns of transformations. Another white box approach for incremental execution of program transformations is based on the concepts of chip and chop proposed by Sittampalam et al. [71]. The proposed approach is, however, concentrated on the incremental execution of transformation of executable specification with an emphasis on behavioral preservation. As such, it is more fitting to the context of code refactoring than that of data model synchronization.

The problem of incremental model synchronization parallels the extensively investigated, yet in some degrees open, problem of view maintenance in databases. Two noteworthy approaches to this problem are presented in [32] and [34]. The view maintenance problem, along with the view update problem, have inspired software engineers to look at interrelated software artifacts as essentially different views of a common database. The bidirectionality of synchronization can thus be remedied by solutions transpired for the view update problem, and incremental synchronization becomes analogous to view maintenance. In spite of similarities, there are also key differences between maintaining database views and synchronization of software models that warrant a distinct research agenda dedicated to the latter. Briefly, database views are defined in few, precise view definition languages, upon which the database community has come to a unanimous consensus. In comparison, model transformation frameworks are rather diverse and immature. Furthermore, the relational database model denotes flat structures. In contrast, models comprise containment hierarchies which pose a semantic for deletion that can be peculiar to handle using purely relational techniques.

Finally, CLIME [64] and MView [33] are constraint based consistency management

frameworks for incremental maintenance of software artifacts. These frameworks rely on the existence of a well-defined set of constraints for ensuring the consistency of the interlinked models, and are capable of incremental resolution of inconsistencies for such cases.

The general methodology as to how interdependent models should be incrementally synchronized is outlined in [40]. The proposed approach, however, requires tight integration with a specific modeling technology, and needs detail knowledge of transformations.

2.3.2 Model Consistency and Dependency

Model Dependency is the problem of specifying and analyzing dependencies between model elements either within the scope of one model (Intra model), or across the boundaries of two inter-related and possibly heterogeneous models (Inter model).

In MDSE, developers use different models for specifying different aspects of a system hence the dispersion of information across multiple documents. It is important to assess the consistency of different but inter-related models to ensure accuracy of models and prevent occurrence of conflicting patterns and behavior discrepancy in the system. Model consistency is the area of research in modeling that deals with identification, formalisation and specification of consistency rules and also provides verification and reconciliation mechanisms to ensure the satisfaction of such constraints.

Description Logics [14] is used as a notation for representing consistency rules between models [73] [72] [43]. Keefe used Dynamic Logic, an extension of model logic intended for reasoning on dynamic behavior of computer programs, as basis of a framework for model consistency [57]. Dynamic Logic broadens model logic with two special model operators that denote post-occurrence necessary and possible predicates for actions. Utilizing these operators, Keefe is able to establish consistency rules between UML class, sequence and state chart diagrams and diagnose inconsistencies by logical inference.

Fombelle et al. in [22] argue that pure active consistency management by monitoring at editing time tend to enforce consistency rules stringently and hence too prohibitive to let developers explore all possibilities which often require tolerating tentative inconsistencies. Instead, they adopt a hybrid approach that manages inconsistencies by defining a set of automata. Collectively, the automatons embroil the state space of models into states that capture evolving model parts' consistency or lack there of. Transitions between the

states assert whether an inconsistency is being introduced, resolved or the current status is retained. They exemplified their approach by a scenario that involves class, state chart and sequence diagrams.

Sabetzadeh and Easterbrook in [66] proposed an approach for analyzing inconsistency of fuzzy viewpoints. Their framework is based on fuzzy set theory to model the viewpoints. They demonstrate that fuzzy viewpoints and fuzzy viewpoints morphisms constitute a *finitely co-complete* category which as a corollary entails that the objects of a category can be completely interconnected without the need for any additional gluing structures. they provide an abstract framework for analyzing inconsistencies during merging incomplete and inconsistent fuzzy viewpoints.

Another approach, proposed by Blanc et al., towards detecting model inconsistencies is to represent models by a sequence of constructive operations that incrementally build the model [13]. This view of models is imperative in nature in contrast to declarative representations conventionally formalized by sets and graph. Structural consistency rules are expressed as logical constraints on the sequences of constructive operations.

A powerful technique for detection and resolution of inconsistency conflicts in graph transformations is *critical pair analysis* [25] and [3]. Mens et al. have done extensive research work capitalizing on this technique to manage inconsistencies during model refactoring and model transformation scenarios [53], [54], [51] and [52]

Formal Concept Analysis is non-deterministic technique which used for the identification of dependency links between models of different levels of abstraction [30] and [27]. Ivkovic and Kotogiannis have adopted this sort of analysis to automatically establish dependency links between PIM and PSM models in a setting purported to the development of commerce applications [39].

Ensuring consistency of models and systems after modification, propagating changes to other software artifacts and model synchronization are overlapping areas of research that are progressively receiving more attention. These problems have been considered before in the context of traditional programming environments as well as data-base systems [35, 64, 33, 17, 1, 34].

2.4 Model Refactoring

Refactorings are behavior preserving modifications intended to improve the internal structure and design of the system without altering its observable interface. Refactoring for models is an important research area that aims to firstly provide similar capabilities of code refactoring facilities that programmers currently enjoy for model developers and also investigate novel opportunities for refactoring operations specific to modeling.

Correa and Werner investigate the common problems in OCL annotated UML models in [19]. They define a collection of OCL smells, analogous to the more notorious code smells blueprints and proceed to offer two categories of refactoring based solutions to resolve these culprit patterns. The first category is specific to OCL and the second one is UML diagram exclusive. The paper gives an outline for the possibility of automating the proposed refactoring in a UML tool. To carry out such refactorings, the OCL expressions have to be parsed as an instance of the OCL metamodel defined by OMG. The paper proposes to use an OCL-like language to perform the updates on the models but inasmuch as OCL is a sans side-effect language, it cannot be used to update graph instances per se. Thus they propose a modified metamodel for OCL to simply tree traversals and define an imperative scripting language that utilizes this modified OCL for queries.

Other works tackling various aspects of model refactoring include [47], [48], [49], [73], [53], [77].

2.5 Partial Evaluation

There exist a vast body of research on partial evaluation. An excellent introductory resource is the book authored by Jones et al. [42], which also provides an exhaustive list of references to the existing literature. Shorter entry point to the area of partial evaluation can be found in [41] and [18]. Sundaresh et al. [74] were amongst the first researchers to exploit partial evaluation for deriving incremental programs, albeit in the context of code-driven programming languages. The indexing model employed in [58] is similar to the scheme we have used for accessing the elements of cached collections. Most of the framework however focus on programming languages; we are not aware of any other research work that leverages partial evaluation techniques in the context of model transformation

languages. As we discuss in Section 5.5, there also tend to be differences in the specialization model transformations, that rely extensively on collection manipulation, and ordinary programs, wherein collection-based caching strategies are not as predominant.

Chapter 3

Characterization of Change

3.1 Model Elements, Models and Metamodels

In a Model Driven Development Environment, every artifact is a model described using a metamodeling language. We adopt the notation $\mathsf{M}_\mathfrak{d} : \mathsf{MM}_\mathfrak{d}$ to express that model $\mathsf{M}_\mathfrak{d}$ is an instance of metamodel $\mathsf{MM}_\mathfrak{d}$ in the context of domain $\mathfrak{d}$. Intuitively, a model is a set of model elements.

For our purpose, model elements are instances of the types defined in the metamodel of a model. In essence, model elements are information bearing entities that enclose primitive and complex information in their *attributes*. They can also have several *containers*, each containing other model elements. Model elements can also *reference* other model elements. The following definition captures these characteristics.

Definition 1 (Model Element) *Model element* $\mathfrak{m}$ *is a tuple* $(\mathfrak{C}, \mathsf{A}, \mathsf{R}, \mathsf{T})$*, in which* $\mathfrak{C}$ *is the list (i.e., a strictly ordered multi-set) of containers,* A *is the list of attribute values,* R *is the list of references,* T *is a mapping to the metamodel, which ascribes a type to the model element.*

We use the same letters as function names for the projection of individual components of model elements, that is to say, $\mathfrak{m} = (\mathfrak{C}(\mathfrak{m}), \mathsf{A}(\mathfrak{m}), \mathsf{R}(\mathfrak{m}), \mathsf{T}(\mathfrak{m}))$. It should be noted that other typical features of models such as names, types and constraints of the attributes and

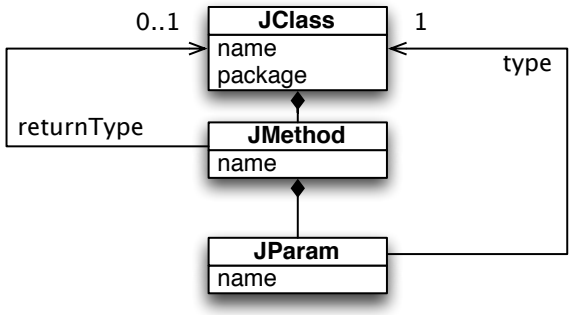


Figure 3.1: Sample Model Abstracting Java Code

containers are all specified in the metamodel and not in the model element itself. This separation allows for type (and also name) agnostic manipulation of model elements.

Definition 1 is recursive; for the members of containment lists are assumed to be model elements themselves. We unify the definition of *model*s and that of *model element*s by representing a model as a model element whose type denotes its metamodel. Differently put, models are regarded as root-level model elements, with only one container which contains all the top-level model elements.

Models are represented in various ways depending on the particular aspect of them that needs to be highlighted. In this chapter, we offer a two-fold representation of models and model elements, to wit, *compositional* and *referential*. These two views are complementary, and to fully specify the semantics of models, they should both be considered along side one another. Nevertheless, decoupling the two views significantly facilitates the analysis and formal reasoning of the algorithms we discuss in this chapter.

There also exists a pragmatic reason for imposing an order on the containers of a model elements. For many change management operations, it is essential to be able to efficiently tell apart two different models, and systematically characterize the differences by a list of edit operations. The general tree-alignment and tree-edit problem, for unordered-trees, has shown to be NP-Hard [11], but is tractable if the tree is ordered.

Figure 3.1 illustrates a sample model, which is an abstraction of Java code in the UML notation. The top-level element represents a Java class. Each class can contain a number of methods, each of which may contain a number of arguments. Furthermore, there are

two kind of references in the model: one is returnType which designates the return type of a method, and the other one is type signifying the type of a method parameter. Using Definition 1, the Java model of Figure 3.1 is as follows.

$$\begin{aligned}
&\mathsf{c1} = (\langle\langle \mathsf{m1} \rangle\rangle, \langle \text{"}\mathsf{SrvImpl}\text{"} \rangle, \varnothing, \mathsf{JClass}) \\
&\mathsf{m1} = (\langle\langle \mathsf{p1} \rangle\rangle, \langle \text{"}\mathsf{method}\text{"} \rangle, \langle \mathsf{String} \rangle, \mathsf{JMethod}) \\
&\mathsf{p1} = (\varnothing, \langle \text{"}\mathsf{str}\text{"} \rangle, \langle \mathsf{String} \rangle, \mathsf{JParam})
\end{aligned}$$

In this set of definitions, $\mathfrak{c}1$ is specified as a class that contains method $\mathfrak{m}1$. The second element of the tuple, i.e., $\text{"}\mathfrak{Srv}\mathfrak{Impl}\text{"}$ is the value of the name attribute. In the definition of method $\mathfrak{m}1$ (and similarly parameter $\mathfrak{p}1$) the third element of the tuple denotes the references specifying the return type of methods (or the type of parameters). For the case of $\mathfrak{m}1$ and $\mathfrak{p}1$, the reference refers to a primitive type, $\mathfrak{String}$.

As noted, information such as attributes and containers' names as well as their types, can be obtained from metamodels. The definition we offered for models, however, is only concerned with instances. For our purpose, we use the following definition for metamodels.

Definition 2 (MetaModel) *A metamodel is a 3-tuple,* $(\mathtt{Sig}_\mathsf{C}, \mathtt{Sig}_\mathsf{A}, \mathtt{Sig}_\mathsf{R})$ *consisting of three signature functions* $\mathtt{Sig}_\mathsf{C} : \mathbb{N} \to \Sigma^* \times \mathcal{T}$, $\mathtt{Sig}_\mathsf{A} : \mathbb{N} \to \Sigma^* \times \mathcal{T}$ *and* $\mathtt{Sig}_\mathsf{R} : \mathbb{N} \to \Sigma^* \times \mathcal{T}$, *which respectively map each container, attribute and reference in the instance model to a tuple comprising its name and type.*

In the definition above, the first argument of each signature function is the index of the attribute for which the meta-information (i.e., name and type) is inquired. The functions return a tuple respectively consisting of a name in Σ^* and a type in $\mathcal{T}$, that is, the set of all types. For example, the following denotes the metamodel of Java code presented earlier in Figure 3.1.

$$\begin{aligned}
&\mathsf{Class} = (\{(1, (\text{"}\mathsf{methods}\text{"}, \mathsf{JMethod}))\}, \{(1, (\text{"}\mathsf{name}\text{"}, \mathsf{String}))\}, \varnothing) \\
&\mathsf{Method} = (\{(1, (\text{"}\mathsf{params}\text{"}, \mathsf{JParam}))\}, \{(1, (\text{"}\mathsf{name}\text{"}, \mathsf{String}))\}, \{(1, (\text{"}\mathsf{returnType}\text{"}, \mathsf{JClass}))\}) \\
&\mathsf{Param} = (\varnothing, \{(1, (\text{"}\mathsf{name}\text{"}, \mathsf{String}))\}, \{(1, (\text{"}\mathsf{type}\text{"}, \mathsf{JClass}))\})
\end{aligned}$$

We define the following auxiliary operators on model elements: $\bigcup\mathfrak{C}$ is a union of all members of $\mathfrak{C}$ which in Definition 1 is defined as a nested list. $\uplus\mathfrak{m}$ returns a flattened list

that contains all the elements that are directly or indirectly contained by model element $\mathfrak{m}$. Both operators are defined formally in the following. It should be noted that the definition of $\uplus$ is recursive and uses $\bigcup$.

$$\bigcup \mathfrak{C} \stackrel{\text{def}}{=} \bigcup_{C_i \in \mathfrak{C}} C_i$$

$$\text{let} \quad \mathfrak{m} = (\mathfrak{C}, A, R, T), \qquad \uplus \mathfrak{m} \stackrel{\text{def}}{=} \begin{cases} \varnothing & \mathfrak{C} = \varnothing \\ \bigcup\limits_{\mathfrak{m}_i \in \bigcup \mathfrak{C}} (\{\mathfrak{m}_i\} \cup \uplus \mathfrak{m}_i) & \text{otherwise} \end{cases}$$

The semantics of containment for model elements is defined as $\mathfrak{m} \in \uplus M$, needless to say $\mathfrak{m} \in \cup \mathfrak{C}(M)$ implies $\mathfrak{m} \in \uplus M$ but not necessarily the other way around; the latter is *immediate* containment by model (element) M, while the former allows containment by any of M's children as well as itself. The following is an example of these two operators on the example Java model.

$$\bigcup \mathfrak{C}(\mathsf{Class}) = \{\mathsf{Method1}\}$$
$$\uplus \mathsf{Class} = \{\mathsf{Method1}, \mathsf{Param1}\}$$

The compositional view of a model (element) represents how all of its contained elements and, likewise, their own contained elements, are structured. For the sake of precision, we should note that the standard set operations, e.g. union and intersection, when applied to lists, implicitly convert the list to multi-sets, by dropping the order, and thus result in multi-sets.

In the realm of type theory, there are two general approaches toward type checking of objects in a type system[59]. One is the Church style of typing, which gives more priority to typing than does the second style, namely the Curry style. In the Church style, only well-typed statements are considered valid. In contrast, the Curry style conceives type checking as a validation process that weeds out ill-typed statements. In the scope of model driven environments, the Curry style of typing seems to be more appropriate, primarily due to the fact that even loosely-typed models can also be used for communication purposes.

Another advantage of isolating the compositional properties of model elements from their referential aspects is that it provides a semantics for *anonymously* addressing model elements within models. Anonymous addresses are name-agnostic. They refer to model

elements by their structural position in the containment hierarchy of a model, without being tied to the name of the containers and those of its higher-up elements. This form of addressing model elements is particularly important when handling modifications, because unlike name-sensitive addresses, such as URIs and Unix Paths, they are invariant to renames and updates of other elements. To give an analogy, the street address of a building changes as does the name of the street in which it resides, even though the building itself is located at exactly the same geographical coordinates. Latitude and longitude, contrariwise, provide immutable positioning references. Similarly, typical addressing schemes such as *URI* and *XPath* that rely on the name of parent elements for identifying elements are volatile to renaming. By separating referential and structural aspects of models and giving prominence to the latter, we devise an addressing scheme that solely relies on the structural positions of model elements which is invariant to any non-structural changes.

More specifically, to form the anonymous address of an element we define two operators, "/" and ".", which are defined in the following.

Definition 3 (Container Selector) *Operator "/" is a function with the signature:* $\text{"/"} : \mathcal{M} \times \mathbb{N} \rightarrow \mathcal{C}$ *in which* $\mathcal{M}$ *is the set of all model elements,* $\mathcal{C}$ *is the set of all containers,* $\mathbb{N}$ *is the set of positive integers.*

$$(\mathfrak{C}, \mathsf{A}, \mathsf{R}, \mathsf{T})/\mathfrak{i} = \pi_{\mathfrak{i}}(\mathfrak{C})$$

The "/" operator is a binary operator whose first operand is a model element and its second operand is a positive integer. $\mathfrak{m}/\mathfrak{n}$ returns the $\mathfrak{n}_{\mathtt{th}}$ container of model element $\mathfrak{m}$. For convenience, we extend the notation of this (and the following) operators to also allow names, as well as positions, when needed.

Definition 4 (Element Selector) *Operator "." is a function with the signature:* $\text{"."} : \mathcal{C} \times \mathbb{N} \rightarrow \mathcal{M}$ *where* $\mathcal{M}$ *is the set of all model elements,* $\mathcal{C}$ *is the set of all containers,* $\mathbb{N}$ *is the set of positive integers.*

$$\mathsf{C}.\mathsf{j} = \pi_{\mathsf{j}}(\mathsf{C})$$

Definition 5 (Attribute Selector) *Operator "@" is a function with the signature:* $\text{"@"} : \mathcal{M} \times \mathbb{N} \rightarrow \Sigma^*$ *where* $\mathcal{M}$ *is the set of all model elements,* $\mathcal{C}$ *is the set of all containers,* $\mathbb{N}$ *is the set of positive integers.*

$$(\mathfrak{C}, \mathsf{A}, \mathsf{R}, \mathsf{T})@\mathfrak{i} = \pi_{\mathfrak{i}}(\mathsf{A})$$

By cascading the container/element segments formed by these operators, we can devise an anonymous addressing scheme which resemble paths, and allow us to navigate, query and access different model elements, and their properties, in a containment hierarchy. The following grammar specifies the syntax of this addressing scheme using the BNF notation.

Addr	::=	[ID | .] | (/((Container.Element) | ..))*
Cont	::=	ID | Number
Element	::=	ID | Number
Attr	::=	Addr@(ID | Number)

Definition 6 (Parent Operator) *Operator "..″ is a function with the following signature* $\text{".."} : \mathcal{M} \times \mathcal{M} \rightarrow \mathcal{M}$*, where* $\mathcal{M}$ *is the set of all model elements.*

$$\mathfrak{m}..M = \begin{cases} \mathfrak{p} & \mathfrak{m} \in \uplus \mathfrak{p} \wedge \mathfrak{p} \in \uplus M \wedge \forall\ \mathfrak{p}'\ \mathfrak{m} \in \uplus \mathfrak{p}' \Rightarrow \mathfrak{p} = \mathfrak{p}' \\ \bot & \text{otherwise} \end{cases}$$

The parent operator finds the immediate element above its given argument in a containment hierarchy. The second argument, the root of the hierarchy, is often implicit in the context and is omitted for the sake of brevity.

Definition 7 (Address) *The (model specific) address of an element is defined as follows.*

$$\text{Addr}_M(M/\alpha) = \alpha$$

For example, the address of `p1` in the previous example is /1.1/1.1 which reads as: *element zero* (i.e., `p1`) *of container zero of element zero* (i.e., `m1`) *of container zero of the root element* (i.e., `c1`). This definition implies that $\mathfrak{m} = M/\text{Addr}_M(\mathfrak{m})$.

3.1.1 Equivalence of two model elements

Equivalence of two model elements asserts that they must have the same types, same attribute values, same references and must also contain *equivalent* child elements, stationed identically in the structure of both models' containment hierarchies. The given definition for the equivalence of two model elements is recursive, for its last criterion in Definition 8,

namely, the equivalence of container lists, requires the equivalence of their members which are again model elements; hence recursively invoking the same equivalence relation of Definition 8.

Definition 8 (Equivalence) *(Deep) Equivalence relation of two model elements* $\mathfrak{m}_1 = (\mathfrak{C}_1, \mathsf{A}_1, \mathsf{R}_1, \mathsf{T}_1)$ *and* $\mathfrak{m}_2 = (\mathfrak{C}_2, \mathsf{A}_2, \mathsf{R}_2, \mathsf{T}_2)$ *is defined as:*

$$\mathfrak{m}_1 = \mathfrak{m}_2 \iff \begin{cases} \mathsf{T}_1 = \mathsf{T}_2 \\ \mathsf{A}_1 = \mathsf{A}_2 \\ \mathsf{R}_1 = \mathsf{R}_2 \\ \mathfrak{C}_1 = \mathfrak{C}_2 \end{cases}$$

As an example, consider the following model definition, alongside the definition of c1 previously presented.

$$\begin{aligned} &\mathsf{c1} = (\langle\langle \mathsf{m1} \rangle\rangle, \langle \text{"SrvImpl"} \rangle, \varnothing, \mathsf{JClass}) \\ &\mathsf{m1} = (\langle\langle \mathsf{p1} \rangle\rangle, \langle \text{"method"} \rangle, \langle \mathsf{String} \rangle, \mathsf{JMethod}) \\ &\mathsf{p1} = (\varnothing, \langle \text{"str"} \rangle, \langle \mathsf{String} \rangle, \mathsf{JParam}) \end{aligned}$$

$$\begin{aligned} &\mathsf{c2} = (\langle\langle \mathsf{m2} \rangle\rangle, \langle \text{"SrvImpl"} \rangle, \varnothing, \mathsf{JClass}) \\ &\mathsf{c3} = (\langle\langle \mathsf{m3} \rangle\rangle, \langle \text{"SrvImpl"} \rangle, \varnothing, \mathsf{JClass}) \\ &\mathsf{m2} = (\langle\langle \mathsf{p2} \rangle\rangle, \langle \text{"method"} \rangle, \langle \mathsf{String} \rangle, \mathsf{JMethod}) \\ &\mathsf{p2} = (\varnothing, \langle \text{"str"} \rangle, \langle \mathsf{String} \rangle, \mathsf{JParam}) \\ &\mathsf{m3} = (\langle\langle \mathsf{p3}, \mathsf{p4} \rangle\rangle, \langle \text{"method"} \rangle, \langle \mathsf{String} \rangle, \mathsf{JMethod}) \\ &\mathsf{p3} = (\varnothing, \langle \text{"num"} \rangle, \langle \mathsf{Integer} \rangle, \mathsf{JParam}) \\ &\mathsf{p4} = (\varnothing, \langle \text{"str"} \rangle, \langle \mathsf{String} \rangle, \mathsf{JParam}) \end{aligned}$$

Based on the definition of the equivalence relation of two model elements, $\mathsf{m1} = \mathsf{m2}$, but $\mathsf{m1} \neq \mathsf{m3}$ because of the discrepancy in their name attributes and the extra parameter the latter possesses. Consequently, $\mathsf{c1} \neq \mathsf{c3}$ due to the inequality of m1 and m3. The equivalence relation of two model elements compares the two elements out of the context of a containing model. To account for the containers and the elements' positions inside them, we augment the equivalence relation with an extra condition which requires the

two model elements have the exact same structural positions in the models in which they reside: model elements $\mathfrak{m}_1$ and $\mathfrak{m}_2$ are held to be *congruent* with respect to models M_1 and M_2 according to the following definition.

Definition 9 (Congruence) *The congruence relation of two model elements* $\mathfrak{m}_1 = (\mathfrak{C}_1, \mathsf{A}_1, \mathsf{R}_1, \mathsf{T}_1)$ *and* $\mathfrak{m}_2 = (\mathfrak{C}_2, \mathsf{A}_2, \mathsf{R}_2, \mathsf{T}_2)$ *with respect to models* M_1 *and* M_2 *is defined as:*

$$\mathfrak{m}_1 \overset{\mathsf{M}_1,\mathsf{M}_2}{\equiv\!\equiv} \mathfrak{m}_2 \iff \begin{cases} \mathfrak{m}_1 = \mathfrak{m}_2 \\ \mathsf{Addr}_{\mathsf{M}_1}(\mathfrak{m}_1) = \mathsf{Addr}_{\mathsf{M}_2}(\mathfrak{m}_2) \end{cases}$$

In the previous example, $\mathsf{p1} = \mathsf{p2} = \mathsf{p4}$. However, $\mathsf{p1} \overset{\mathsf{c1},\mathsf{c2}}{\equiv\!\equiv} \mathsf{p2}$ but $\mathsf{p1} \overset{\mathsf{c1},\mathsf{c3}}{\not\equiv\!\equiv} \mathsf{p4}$, because $\mathsf{Addr}_{\mathsf{c1}}(\mathsf{p1}) = /1.1/1.1$, whereas $\mathsf{Addr}_{\mathsf{c3}}(\mathsf{p4}) = /1.1/1.2$.

It is also possible to compare two model elements only with respect to the number of their direct children and their attribute values and references, that is to say, to ignore discrepancies between their non-immediate descendant elements. The notion of weak equivalence is defined in the following to allow for such comparisons.

Definition 10 (Weak Equivalence) *The weak equivalence relation of two model elements* $\mathfrak{m}_1 = (\mathfrak{C}_1, \mathsf{A}_1, \mathsf{R}_1, \mathsf{T}_1)$ *and* $\mathfrak{m}_2 = (\mathfrak{C}_2, \mathsf{A}_2, \mathsf{R}_2, \mathsf{T}_2)$ *is defined as follows:*

$$\mathfrak{m}_1 \cong \mathfrak{m}_2 \iff \begin{cases} \mathsf{T}_1 = \mathsf{T}_2 \\ \mathsf{A}_1 = \mathsf{A}_2 \\ \mathsf{R}_1 = \mathsf{R}_2 \\ \forall \mathfrak{i} \quad |\pi_{\mathfrak{i}}(\mathfrak{C}_1)| = |\pi_{\mathfrak{i}}(\mathfrak{C}_2)| \end{cases}$$

Weak equivalence is an equivalence relation inasmuch as it is manifestly reflexive, transitive and symmetric. It is primarily defined to only hold two elements to be identical, notwithstanding the differences down in the containment hierarchy or lack thereof. The primarily motivation for defining the weak equivalence relation is to confine the impact set of a change, that is, to avoid including all the elements in the containment path of a changed element up to the root of the containment hierarchy. For example, even though $\mathsf{c1} \neq \mathsf{c3}$, they are weakly equivalent, since comparing only $\mathsf{m1}$ with $\mathsf{m3}$ notwithstanding their children (i.e., parameters) results in no difference.

3.2 Atomic Model Manipulation Operators

We adopt a compositional representation of change in models. That is to say, every change operation can be expressed as a composition of finer grained change operators. This requires identifying a number of *atomic* change operations as the basis of the space of possible changes. We introduce five atomic change operations that take part in change composition. The notations used for each of the atomic changes are listed as follows.

In the following function signatures, $\mathcal{M}$ is the set of all model elements, $\mathcal{T}$ is the set of all types, $\mathcal{C}$ set of all containers, $\mathcal{A}$ set of all attributes, and Σ^* is the set of all attribute values. The purpose of unions is to overload the functions to accept both positions of elements (and containers) as well as their symbolic references in appropriate contexts.

Definition 11 (Update) *The atomic update attribute is a function:*
$\blacklozenge : \mathcal{M} \times (\mathcal{A} \cup \mathbb{N}) \times \Sigma^* \longrightarrow \mathcal{M}$
For any model element $\mathfrak{m} = (\mathfrak{C}, \langle \mathfrak{a}_1, \ldots, \mathfrak{a}_{i-1}, \mathfrak{a}_i, \mathfrak{a}_{i+1}, \ldots, \mathfrak{a}_{|A|} \rangle, R, T)$

$$\blacklozenge(\mathfrak{m}, i, \nu) = (\mathfrak{C}, \langle \mathfrak{a}_1, \ldots, \mathfrak{a}_{i-1}, \nu, \mathfrak{a}_{i+1}, \ldots, \mathfrak{a}_{|A|} \rangle, R, T)$$

The update function returns a model element that is identical to $\mathfrak{m}$ except for its i_{th} attribute whose value is ν.

Definition 12 (Insertion) *The atomic element insertion is a function:*
$\blacktriangle : \mathcal{M} \times \mathcal{T} \times (\mathcal{C} \cup \mathbb{N}) \longrightarrow \mathcal{M}$
For any model element $\mathfrak{m} = (\mathfrak{C}, A, R, T)$
in which $\mathfrak{C} = \langle C_1, \ldots, C_{i-1}, \langle \mathfrak{m}_1, \ldots, \mathfrak{m}_{|C_i|} \rangle, C_{i+1}, \ldots, C_{|\mathfrak{C}|} \rangle$

$$\blacktriangle(\mathfrak{m}, T', i) = (\mathfrak{C}', A, R, T)$$

where $\mathfrak{C}' = \langle C_1, \ldots, C_{i-1}, \langle \mathfrak{m}_1, \ldots, \mathfrak{m}_{|C_i|}, \mathfrak{m}' \rangle, C_{i+1}, \ldots, C_{|\mathfrak{C}|} \rangle$ *and*

$$\mathfrak{m}' = (\langle \overbrace{\varnothing, \ldots, \varnothing}^{|\mathfrak{C}_{T'}| \text{ times}} \rangle, \langle \overbrace{\perp, \ldots, \perp}^{|A_{T'}| \text{ times}} \rangle, \langle \overbrace{\varnothing, \ldots, \varnothing}^{|R_{T'}| \text{ times}} \rangle, T')$$

$|A_{T'}|$, $|R_{T'}|$ and $|\mathfrak{C}_{T'}|$ are the number of attributes, the number of reference groups and number of containers of type T', respectively. The atomic element insertion function returns

a model element that is identical to $\mathfrak{m}$ in type, attributes, references and all container but the $\mathfrak{i}_{\mathfrak{th}}$ one which is appended with a new raw model element of type T'.

Definition 13 (Deletion) *The atomic element deletion is a function:*
$\blacktriangledown : \mathcal{M} \times (\mathcal{C} \cup \mathbb{N}) \longrightarrow \mathcal{M}$
For any model element $\mathfrak{m} = (\mathfrak{C}, \mathsf{A}, \mathsf{R}, \mathsf{T})$
in which $\mathfrak{C} = \langle \mathsf{C}_1, \ldots, \langle \mathfrak{m}_1, \ldots, \mathfrak{m}_{|\mathsf{C}_i|-1}, \mathfrak{m}_{|\mathsf{C}_i|} \rangle, \mathsf{C}_{i+1}, \ldots, \mathsf{C}_{|\mathfrak{C}|} \rangle$

$$\blacktriangledown(\mathfrak{m}, \mathfrak{i}) = (\langle \mathsf{C}_1, \ldots, \langle \mathfrak{m}_1, \ldots, \mathfrak{m}_{|\mathsf{C}_i|-1} \rangle, \mathsf{C}_{i+1}, \ldots, \mathsf{C}_{|\mathfrak{C}|} \rangle, \mathsf{A}, \mathsf{R}, \mathsf{T})$$

The atomic delete function purges the last element of the $\mathfrak{i}_{\mathfrak{th}}$ container of its input model element.

Definition 14 *The atomic reference insertion is a function:*
$\triangle \colon \mathcal{M} \times \mathcal{ID}_{\mathcal{M}} \longrightarrow \mathcal{M}$
For any model element $\mathfrak{m} = (\mathfrak{C}, \mathsf{A}, \mathsf{R}, \mathsf{T})$
in which

$$\triangle\ (\mathfrak{m}, \mathfrak{r}) = (\mathfrak{C}, \mathsf{A}, \mathsf{R} \uplus \langle \mathfrak{r} \rangle, \mathsf{T})$$

The atomic reference insertion function returns a model element that is identical to $\mathfrak{m}$ in type, attributes, containers but it has an extra reference $\mathfrak{r}$ appended to its otherwise identical list of references (i.e., R).

Definition 15 *The atomic reference deletion is a function:*
$\triangledown : \mathcal{M} \longrightarrow \mathcal{M}$

$$\triangledown((\mathfrak{C}, \mathsf{A}, \langle \mathfrak{r}_1, .., \mathfrak{r}_n \rangle, \mathsf{T})) = (\mathfrak{C}, \mathsf{A}, \langle \mathfrak{r}_1, .., \mathfrak{r}_{n-1} \rangle, \mathsf{T})$$

The atomic reference deletion function purges the last referenced element of its input model element.

The semantics of composition is also similar to that of mathematical functions. For example, the following composite change operation adds a new parameter to the method of the Java example, and then, updates its name to *"param2"*:

$$\blacklozenge(\blacktriangle(\mathsf{m1}, \mathsf{params}, \mathsf{JParam}), \mathsf{name}, \text{"}\mathsf{param2}\text{"})$$

The same operation can also be expressed using the anonymous scheme:

$$\blacklozenge(\blacktriangle(/1.1, 1, \mathsf{JParam}), 1, \text{"}\mathsf{param2}\text{"})$$

We adopt the following syntactic sugar for updating attributes: $\blacklozenge(\mathfrak{m}@\mathfrak{a} \mapsto \nu) = \blacklozenge(\mathfrak{m}, \mathfrak{i}_\mathfrak{a}, \nu)$, where $\mathfrak{i}_\mathfrak{a}$ is the index of attribute $\mathfrak{a}$ taken from the type of $\mathfrak{m}$.

3.3 Change Factorization and Model Edit Distance

As mentioned in Chapter 2, there are two possible perspectives toward changing models, namely, state-based and operation-based views of change. Change factorization is based on tree edit-distance and tree alignment problems. The original treatment of these problems are due to Tai [75]. We, present here a series of algorithms that, given two models, calculate a sequence of atomic operations that converts the first model to the second one. The returned sequence has minimum edit cost, with respect to some optimization criteria. This is similar to the edit distance of two strings, only it is applied on model elements.

Lemma 3.3.1 (Completeness of Atomic Changes) *For any two given models* $\mathfrak{m}_1$ *and* $\mathfrak{m}_2$ *of the same type* T*, there exists a sequence of change* $\delta^* = \delta_1 \circ \delta_2 \circ \ldots \circ \delta_\mathfrak{n}$ *such that* $\delta^*(\mathfrak{m}_1) = \mathfrak{m}_2$.

Proof. There is a trivial change sequence that first reduces $\mathfrak{m}_1$ to $(\varnothing, \langle \bot, .., \bot \rangle, \varnothing, \mathsf{T})$ and from there it constructs $\mathfrak{m}_2$ element by element.

□

A factor that influences the process of change factorization is the precise semantics chosen for change operations, in particular the structural ones. If we allow insertion and deletion operations to only affect the last element of each container, we end up with a different set of operations than when we allow insertion and deletion at arbitrary locations in any container. We investigate both cases in the following.

Change factorization using $\blacklozenge(\mathfrak{m}, \mathfrak{i}, \nu)$, $\blacktriangle(\mathfrak{m}/\mathfrak{i})$, $\blacktriangledown(\mathfrak{m}/\mathfrak{i})$
The introduced insertion and deletion operations in Section 3.2, are not able to insert to and delete from an arbitrary position in a container. They are designed so that their application would preserve the structural address of the remaining elements of a model.

Another advantage of adopting these as the primitive structural change operations is that an algorithm with linear time and constant space complexity exists to factorize any two given models into a composition of these changes, as listed in Algorithm 1. The essence of this algorithm is that first tries to make the two models equivalent up to isomorphism (i.e., having no structural differences), and then apply update operations to make them have the same attribute values. This algorithm effectively associates zero cost to update and non-zero costs to insertion and deletion. In the pseudo code (listed in Algorithm 1), lines 2-6 updates the root element of M to match that of N. The algorithm then proceeds to streamline the children of the roots of M and N by recursion. In lines 11 to 14 it recursively calls itself to align each existing element. If M has more elements in a container than N does in the corresponding container, the algorithm deletes the excess elements from M (lines 21-25). Otherwise, it inserts new elements and updates them to match the extra elements in N (lines 15-20).

The algorithm recursively traverses the containment hierarchy of both model elements, visiting each element only once. The runtime cost of updating an element is constant therefore the algorithm has the worst case runtime complexity of $O(\mathtt{max}(|\uplus \mathrm{M}|, |\uplus \mathrm{N}|))$. Since no memoization, other than the local variables, are required, a judicious implementation can achieve constant space complexity.

Change factorization using $\blacklozenge(\mathtt{m}, \mathtt{i}, \mathtt{v})$, $\blacktriangle(\mathtt{m}/\mathtt{i}.\mathtt{j})$, $\blacktriangledown(\mathtt{m}/\mathtt{i}.\mathtt{j})$

Algorithm 1 can yield to quite inexorable edit sequences due to the limitation that elements can only be appended or dropped to/from containers. Although the algorithm produces minimum-cost edit scripts with respect to these given change operations, shorter edit sequences can be achieved if we lift these constraints. Assuming that model elements can be inserted to and delete from arbitrary positions of containers, we can achieve simpler and more pragmatic edit scripts.

String edit distance is a classic problem to align two arbitrary sequences of characters by applying a series of change operations comprising updating a character, inserting a new character and deleting a character to one sequence so as to make it match the other one[69]. The classic solution is based on dynamic programming and has the run-time complexity of $O(\mathtt{m} \cdot \mathtt{n})$, where $\mathtt{m}$ and $\mathtt{n}$ are the sizes the two sequences. A similar solution for model elements with multiple containers is presented in Algorithm 2. In Algorithm 2 the $\mathtt{last}$ function returns the last container of a model element. γ associates a cost to each change

Algorithm 1 Change Factorization with append and drop

function $\mathtt{factorize}(\mathsf{M}, \mathsf{N}, \Delta)$
  for $\mathfrak{i} \leftarrow 1..|\mathsf{A}(\mathsf{M})|$ **do**
    if $\pi_{\mathfrak{i}}(\mathsf{A}^{\mathsf{M}}) \neq \pi_{\mathfrak{i}}(\mathsf{A}^{\mathsf{N}})$ **then**
      $\Delta \leftarrow \Delta \uplus \blacklozenge(\mathsf{M}, \mathfrak{i}, \pi_{\mathfrak{i}}(\mathsf{A}^{\mathsf{N}}))$
    end if
  end for
  for $\mathfrak{C}^{\mathsf{M}}_{\mathfrak{i}} = \pi_{\mathfrak{i}}(\mathfrak{C}^{\mathsf{M}}),\ \mathfrak{C}^{\mathsf{N}}_{\mathfrak{i}} = \pi_{\mathfrak{i}}(\mathfrak{C}^{\mathsf{N}}), \mathfrak{i} \leftarrow 1..|\mathfrak{C}^{\mathsf{M}}|$ **do**
    $\mathfrak{j} \leftarrow 0$
    $\mathfrak{m} \leftarrow |\mathfrak{C}^{\mathsf{M}}_{\mathfrak{i}}|$
    $\mathfrak{n} \leftarrow |\mathfrak{C}^{\mathsf{N}}_{\mathfrak{i}}|$
    while $\mathfrak{j} \leqslant \mathfrak{min}(\mathfrak{m}, \mathfrak{n})$ **do**
      $\Delta \leftarrow \Delta \uplus \mathtt{factorize}(\pi_{\mathfrak{j}}(\mathfrak{C}^{\mathsf{M}}_{\mathfrak{i}}), \pi_{\mathfrak{j}}(\mathfrak{C}^{\mathsf{N}}_{\mathfrak{i}}), \Delta)$
      $\mathfrak{j} \leftarrow \mathfrak{j} + 1$
    end while
    if $\mathfrak{m} < \mathfrak{n}$ **then**
      while $\mathfrak{j} \leqslant \mathfrak{n}$ **do**
        $\Delta \leftarrow \Delta \uplus \blacktriangle(\mathsf{M}/\mathfrak{i})$
        $\Delta \leftarrow \Delta \uplus \mathtt{factorize}((\varnothing, \langle \bot, .., \bot \rangle, \varnothing, \mathtt{Type}(\mathfrak{C}^{\mathsf{M}}_{\mathfrak{i}})), \pi_{\mathfrak{j}}(\mathfrak{C}^{\mathsf{N}}_{\mathfrak{i}}))$
        $\mathfrak{j} \leftarrow \mathfrak{j} + 1$
      end while
    else if $\mathfrak{m} > \mathfrak{n}$ **then**
      while $\mathfrak{j} \leqslant \mathfrak{m}$ **do**
        $\Delta \leftarrow \Delta \uplus \blacktriangledown(\mathsf{M}/\mathfrak{i})$
        $\mathfrak{j} \leftarrow \mathfrak{j} + 1$
      end while
    end if
  end for
  return Δ
end function

operation, and, $\mathfrak{d}(\mathfrak{s},\mathfrak{t})$ is a table that memoizes $\delta(\mathfrak{s},\mathfrak{t})$: the minimum edit distance for converting model element $\mathfrak{s}$ to model element $\mathfrak{t}$ using a combination of update attribute, insertion to, and deletion from an arbitrary point in the container. The following equations highlight the gist of the algorithm.

$$\mathfrak{last}(\mathfrak{C}) = \mathfrak{sup}\{\ \{\mathfrak{i} | \pi_\mathfrak{i}(\mathfrak{C}) \neq \varnothing \wedge \forall\ \mathfrak{j} > \mathfrak{i}\ \ \pi_\mathfrak{j}(\mathfrak{C}) = \varnothing\} \cup \{0\}\ \} \tag{1}$$

$$\gamma(\mathfrak{a},\mathfrak{b}) = \begin{cases} 0 & \mathfrak{a} = \mathfrak{b} \\ 1 & \mathit{otherwise} \end{cases} \tag{2}$$

$$\gamma(\langle \mathfrak{a}_1,..,\mathfrak{a}_n\rangle, \langle \mathfrak{b}_1,..,\mathfrak{b}_n\rangle) = \Sigma_{\mathfrak{i}=1}^{n}\gamma(\mathfrak{a}_\mathfrak{i},\mathfrak{b}_\mathfrak{i}) \tag{3}$$

$$\delta((\varnothing,\mathsf{A}),(\varnothing,\mathsf{B})) = \gamma(\mathsf{A},\mathsf{B}) \tag{4}$$

$$\delta(\mathfrak{s} = (\mathfrak{C},\mathsf{A}), \mathfrak{t} = (\varnothing,\mathsf{B})) = \gamma(\blacktriangledown) + \delta(\blacktriangledown(\mathfrak{s}/1),\mathfrak{t}) \tag{5}$$

$$\delta(\mathfrak{s} = (\varnothing,\mathsf{A}), \mathfrak{t} = (\mathfrak{D},\mathsf{B})) = \gamma(\blacktriangle) + \delta(\mathfrak{s},\blacktriangledown(\mathfrak{t}/1)) + \delta(\varnothing,\mathfrak{t}/\mathfrak{last}(\mathfrak{D}).|\mathfrak{D}_{\mathfrak{last}(\mathfrak{D})}|) \tag{6}$$

$$\delta(\mathfrak{s} = (\mathfrak{C},\mathsf{A}), \mathfrak{t} = (\mathfrak{D},\mathsf{B})) = \min \begin{cases} \delta(\blacktriangledown(\mathfrak{s}/\mathfrak{last}(\mathfrak{C})),\blacktriangledown(\mathfrak{t}/\mathfrak{last}(\mathfrak{D}))) + \delta(\mathfrak{s}/\mathfrak{last}(\mathfrak{C}).|\mathfrak{C}_{\mathfrak{last}(\mathfrak{C})}|, \mathfrak{t}/\mathfrak{last}(\mathfrak{D}).|\mathfrak{D}_{\mathfrak{last}(\mathfrak{D})}|) \\ \delta(\mathfrak{s},\blacktriangledown(\mathfrak{t}/\mathfrak{last}(\mathfrak{D}))) + \delta((\varnothing,\varnothing),\mathfrak{t}/\mathfrak{last}(\mathfrak{D}).|\mathfrak{D}_{\mathfrak{last}(\mathfrak{D})}|) + \gamma(\blacktriangle) \\ \delta(\blacktriangledown(\mathfrak{s}/\mathfrak{last}(\mathfrak{C})),\mathfrak{t}) + \gamma(\blacktriangledown) \end{cases} \tag{7}$$

Equation 1 defines an auxiliary function, $\mathfrak{last}$, which returns the index of the last non-empty container in a list of containers. It looks for an index after which there is either no container in the list, or all other subsequent containers are empty. For an empty list it returns 0. As mentioned, γ is the cost function. Equation 2 indicates that the cost of matching two attributes is zero if they are equal, or 1 otherwise. The cost function is overloaded for lists of attributes in equation 3. The cost of matching two lists is simply the sum of matching their attributes in identical positions. Equations 4-7 define the edit-distance of two models based on a given cost function. The distance of two model elements that contain no children is defined in equation 4 as the cost of matching their attributes, as defined in equation 3. Equation 5 states the distance between a model element that possibly contains some children nodes and a leaf model element: it is recursively defined as the cost of deleting one element from the source model $\mathfrak{s}$ plus the distance between the resulting model and $\mathfrak{t}$. Similarly, equation 6 recursively defines the opposite case by an insertion operation. Equation 7 defines the generic distance function for arbitrary model

elements: at each step of recursion, one of the three operations which minimizes the total distance is chosen. Repeated expansion of equation 7, along with the base case equations 4-6, allows for calculating the minimum edit distance of two models.

Theorem 3.3.1 *Algorithm 2 has runtime and space complexity of*

$$O(\sum_{k=0}^{h_{max}} \sum_{i} |s_k/i||t_k/i| \quad)$$

Where $h_{max} = max\{depth(S), depth(T)\}$ *and* s_k *and* t_k *are the* k_{th} *descendant elements of* S *and* T *respectively.*

Proof. Effectively, the algorithm aligns the elements of each container with its counterpart in the changed model. Assume that each container has a special null value. Aligning each element on Ss side with this null value is tantamount to deleting that element. Similarly, aligning the null element of S with each element of T signifies the insertion of a new element to the container. Any other alignment pertains to updating the aligned element of S to match an element in T. If the container i of s has m elements, and that of t has n elements, there exist $(m + 1)(n + 1)$ alignments for the whole container. As explicated in the last equation above (and on line 26 of Algorithm 1), each alignment leads to at most three subproblems, each of which occupies a constant space in the memorization table and a constant time for calculating its cost. To obtain the total space and time complexity orders we, thus, have to sum this over all the containers of any two corresponding model elements in the corresponding depth level of the containment hierarchies in s and t.

□

Figures 3.2 illustrate the result of experiments that have been done on random trees using Algorithm 2. It is evident from the figure that both measured size (represented by $|d|$) and elapsed time (represented by T) are asymptotically bound by the equation of Theorem 3.3.1 (represented, using constant factor 3, in the figure by $O(d)$).

In Algorithmalg:factorize2, the $c(S, T)$ matrix holds the actual change selected as the optimized edit step for models S and T. This is used in conjunction with $d(S, T)$, the table for edit cost of each subproblem, to compute using simple back-tracking the actual change sequence needed to transform S to T.

Algorithm 2 Change Factorization with insert and delete

1: **function** $\delta(S, T)$
2: **if** $(\text{cached} \leftarrow d(\langle S, T\rangle)) \neq \varnothing$ **then**
3: **return** cached
4: **end if**
5: **if** $\mathfrak{C}^S = \mathfrak{C}^T = \varnothing$ **then**
6: $\Delta \leftarrow \varnothing$
7: **for** $i \leftarrow 1..|A^S|$ **do**
8: **if** $\pi_i(A^S) \neq \pi_i(A^T)$ **then**
9: $\Delta = \Delta \uplus \blacklozenge(S, i, \pi_i(A^T))$
10: **end if**
11: **end for**
12: $\mathit{change}(S, T) \leftarrow \Delta$
13: $d(S, T) \leftarrow |\Delta| \times \gamma(\blacklozenge)$
14: **return** $d(S, T)$
15: **end if**
16: **if** $\mathfrak{C}^S = \varnothing$ **then**
17: $\mathit{change}(S, T) \leftarrow \blacktriangle(S/\mathit{last}(\mathfrak{C}^S))$
18: $d(S, T) \leftarrow \gamma(\blacktriangle) + \delta(S, \blacktriangledown(T/\mathit{last}(\mathfrak{C}^T))) + \delta(\varnothing, T/\mathit{last}(\mathfrak{C}^T).|\mathfrak{C}^T_{\mathit{last}(\mathfrak{C}^T)}|)$
19: **return** $d(S, T)$
20: **end if**
21: **if** $\mathfrak{C}^T = \varnothing$ **then**
22: $\mathit{change}(S, T) \leftarrow \blacktriangledown(T/\mathit{last}(\mathfrak{C}^T))$
23: $d(S, T) \leftarrow \gamma(\blacktriangledown)$
24: **return** $d(S, T)$
25: **end if**
26: $d(S, T) \leftarrow \min \begin{cases} \mathit{updateCost} = \delta(\blacktriangledown(S/\mathit{last}(\mathfrak{C}^S)), \blacktriangledown(T/\mathit{last}(\mathfrak{C}^T))) + \\ \delta(S/\mathit{last}(\mathfrak{C}^S).|\mathfrak{C}^S_{\mathit{last}(\mathfrak{C}^S)}|, T/\mathit{last}(\mathfrak{C}^T).|\mathfrak{C}^T_{\mathit{last}(\mathfrak{C}_T)}|) \\ \\ \mathit{insertCost} = \delta(S, \blacktriangledown(T/\mathit{last}(\mathfrak{C}^T))) + \\ \delta((\varnothing, \varnothing), t/\mathit{last}(\mathfrak{C}^T).|\mathfrak{C}^T_{\mathit{last}(\mathfrak{C}^T)}|) + \gamma(\blacktriangle) \\ \\ \mathit{deleteCost} = \delta(\blacktriangledown(S/\mathit{last}(\mathfrak{C}^S)), T) \end{cases}$
27: **if** $d(S, T) = \mathit{updateCost}$ **then**
28: $\mathit{change}(S, T) \leftarrow \blacklozenge$
29: **else if** $d(S, T) = \mathit{insertCost}$ **then**
30: $\mathit{change}(S, T) \leftarrow \blacktriangle$
31: **else**
32: $\mathit{change}(S, T) \leftarrow \blacktriangledown$
33: **end if**
34: **return** $d(S, T)$
35: **end function**

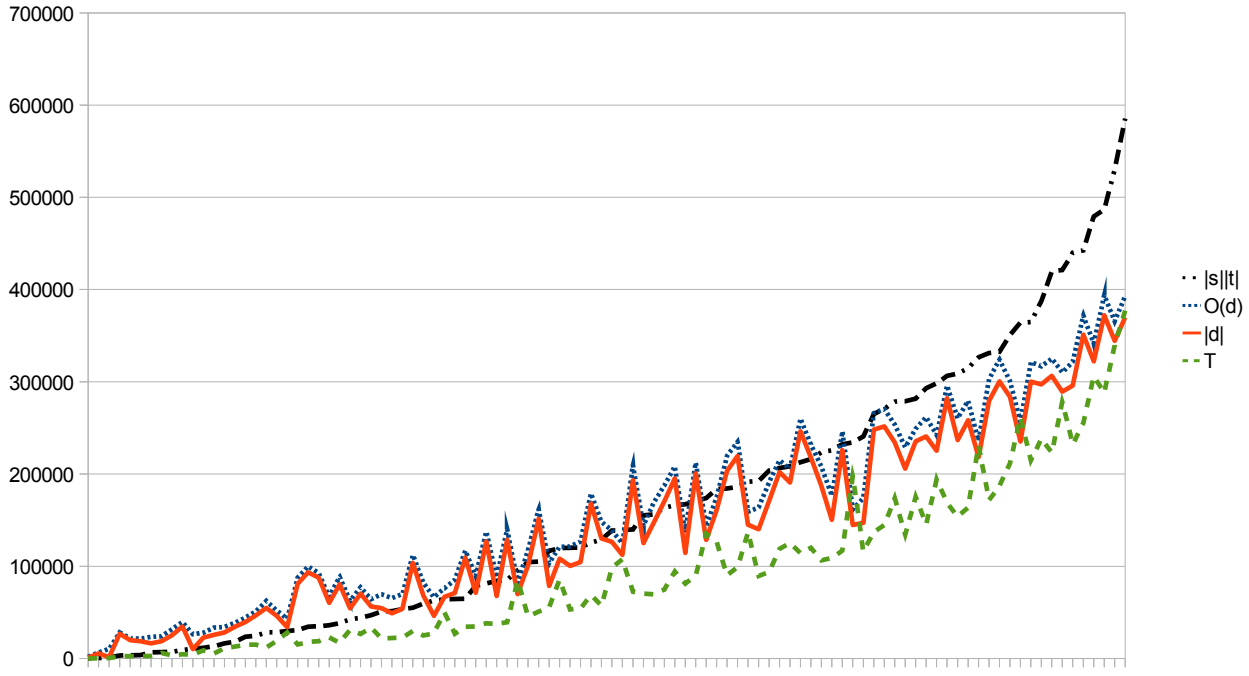


Figure 3.2: Runtime and Space Complexity of Dynamic Programming Change Factorization Algorithm. X-Axis is n, the size of the randomly generated model, and Y-Axis measures the value of $O(d)$ as represented in Theorem 3.3.1 with a constant of 3, the actual size of the table is represented by $|d|$, the product of the size of two models is represented by $|s|.|t|$ for comparison. T is the runtime of the algorithm which is normalized and projected for comparison with the space complexity.

3.3.1 Edit Script Normalization

In this section, we present a set of rules that can be used to normalize edit-sequences to equivalent ones by eliminating idempotent operations and those whose effects cancel one another. These rules should be viewed akin to algebraic rules used for symbolic simplification of algebraic expression. The presented rules are value agnostic; they can be applied before effecting the change to any model to simplify the edit script.

$$
\begin{aligned}
\blacklozenge(s/i.j/x@a \mapsto v); \blacktriangledown(s/i) &= \begin{cases} \blacktriangledown(s/i) & j = |s/i| \\ \blacktriangledown(s/i); \blacklozenge(s/i.j/x@a \mapsto v) & \text{otherwise} \end{cases} \\
\blacklozenge(s/i.j); \blacktriangle(s/i) &= \blacktriangle(s/i); \blacklozenge(s/i.j) \\
\blacklozenge(s/i.j@k \mapsto v); \blacklozenge(s'/i'.j'@k' \mapsto v') &= \begin{cases} \blacklozenge(s/i.j@k \mapsto v) & s = s', i = i', j = j', k = k' \\ \blacklozenge(s'/i'.j'@k' \mapsto v'); \blacklozenge(s/i.j@k \mapsto v) & \text{otherwise} \end{cases} \\
\blacktriangledown(s/i); \blacktriangledown(s'/j) &= \begin{cases} \blacktriangledown(s/i) & s' \in \uplus s/i.\text{last} \\ \blacktriangledown(s'/j) & s \in \uplus s'/j.\text{last} \\ \blacktriangledown(s'/j); \blacktriangledown(s/j) & \text{otherwise} \end{cases} \\
\blacktriangle(s/i); \blacktriangledown(s/i) &= \text{identity} \\
\blacktriangle(s/i.k/a); \blacktriangledown(s/i) &= \text{identity}
\end{aligned}
$$

Proofs of these rules are straight-forward corollaries of the definitions of change operators. The first rule basically states that an update operation is cancelled by the deletion of one of its direct or indirect parents. Otherwise, the two changes are independent and their order can be interchanged. The second rule asserts that an update and an insertion are always independent and can always be interchanged. The third rule states that if two updates target the same attribute of the same element then the update that is applied latest overrides the earlier one. A corollary of this rule is that Update is idempotent. The forth rule highlights the net effect of two delete operations. If either target of the delete element is a descendant of the other operation then only the deletion of the element which is higher in the containment hierarchy is needed due to the fact that it purges the other one as well. Finally, the last two rules express the cancelation of insertion by deletion.

3.3.2 Impact set of a Change

Intuitively, an impact set of a change operation on a model is the set of all elements in the model that would be modified if the change is applied. Formally, we define the Impact Set for an update Δ over model M as:

Definition 16 (Impact Set) *The Impact set of change* $\Delta : \mathcal{L}(MM) \to \mathcal{L}(MM)$ *on model* $M \in \mathcal{L}(MM)$ *is defined as:*

$$\mathcal{I}^{\Delta}_{M} = \{m \in \uplus M |\ \ \Delta(M)/Addr_{M}(m) \ncong m\}$$

The constructive definition denoted above includes all elements inside model M whose positions in $\Delta(M)$, the modified model, is occupied by an unequal (in the weak equivalence sense) element. The expression applies the overloaded "/" operator to refer to the model element in $\Delta(M)$ that resides in the exact anonymous address denoted by $Addr_{M}(m)$, i.e., the address of model element m in model M. Impact set essentially includes the elements of the source model that are modified as a result of applying Δ. Weak equivalence is chosen over the stronger notion, because it is desired to only include the elements that are modified and not the entire upward path to the root of the containment hierarchy. The impact set for the atomic change operations are as follows.

Lemma 3.3.2

$$\mathcal{I}^{\blacklozenge(m,i,v)}_{M} = \begin{cases} \{m\} & m \in \uplus M \wedge A_{m}(i) \neq v \\ \varnothing & otherwise \end{cases}$$

$$\mathcal{I}^{\blacktriangle(m,i)}_{M} = \begin{cases} \{m\} & m \in \uplus M \\ \varnothing & otherwise \end{cases}$$

$$\mathcal{I}^{\blacktriangledown(m,i)}_{M} = \begin{cases} \{m, n\} \cup \uplus m/i.last & m \in \uplus M \wedge m/i \neq \varnothing \\ \varnothing & otherwise \end{cases}$$

Proof. Straightforward from the definitions of change operators and impact set.

□

Lemma 3.3.3 $\forall \Delta \in \{\blacklozenge, \blacktriangle\}^{*}.\ \ m \overset{M,N}{\equiv\equiv} n \ \ \Rightarrow \Delta(M)/Addr_{M}(m) \overset{\Delta(M),\Delta(N)}{\equiv\equiv} \Delta(N)/Addr_{N}(n)$

Proof. The congruence relationship requires the equivalence of addresses in the two models, which easily follows from the premise of the lemma and Definition 7:

$$Addr_{\Delta(M)}(\Delta(M)/Addr_{M}(m)) = Addr_{M}(m) = Addr_{N}(n) = Addr_{\Delta(N)}(\Delta(N)/Addr_{N}(n))$$

To show the equivalence of elements m and n, we use structural induction only on Δs that affect elements m and n, i.e., $m \in \mathcal{I}^{\Delta}_{M}$ and $n \in \mathcal{I}^{\Delta}_{N}$.

- *Base case* There are two cases: $\Delta = \blacklozenge(\alpha)$ and $\Delta = \blacktriangle(\alpha)$. For both cases from $\mathfrak{m} \in \mathcal{I}_M^\Delta$ and Lemma 3.3.2 it follows that $\alpha = \mathtt{Addr}_M(\mathfrak{m})$. From the definitions of $\blacklozenge$ and $\blacktriangle$ it follows that $\Delta(M)/\alpha = \Delta(N)/\alpha$.

- *Induction Step* for $\Delta \in \{\blacklozenge, \blacktriangle\}^*$ we assume that the lemma holds. From the base case it immediately follows that for any $\delta \in \{\blacklozenge, \blacktriangle\}$ we have:

$$\delta(M)/\mathtt{Addr}_M(\mathfrak{m}) \overset{\delta(M),\delta(N)}{\equiv\!\equiv} \delta(N)/\mathtt{Addr}_N(\mathfrak{n})$$

□

Lemma 3.3.3 is not necessarily valid for change sequences that involve deletion. The following is a counter example:

$$\begin{aligned} M &= (\langle\langle \mathfrak{m} \rangle\rangle, \Box, \Box, \Box) \\ N &= (\langle\langle \mathfrak{m}, \mathfrak{n} \rangle\rangle, \Box, \Box, \Box) \\ \Delta &= \blacktriangledown(\Box/1) \\ \Delta(M) &= (\varnothing, \Box, \Box, \Box) \\ \Delta(N) &= (\langle\langle \mathfrak{m} \rangle\rangle, \Box, \Box, \Box) \\ (\Delta(M)/1 = \bot) &\neq (\Delta(N)/1 = \mathfrak{m}) \end{aligned}$$

Lemma 3.3.4 $\forall \Delta \in \{\blacklozenge, \blacktriangle\}^*. \quad \mathfrak{m} \in \uplus M \wedge \mathfrak{n} \in \uplus N \wedge \mathtt{Addr}_M(\mathfrak{m}) = \mathtt{Addr}_N(\mathfrak{n}) \wedge \mathfrak{m} \cong \mathfrak{n} \Rightarrow \Delta(M)/\mathtt{Addr}_M(\mathfrak{m}) \cong \Delta(N)/\mathtt{Addr}_N(\mathfrak{n})$

Proof. Identical to the proof of Lemma 3.3.3 replacing equivalence with weak-equivalence. □

Theorem 3.3.2 *For model* M *and two arbitrary compositions of atomic insertion and update changes* $\Delta_1, \Delta_2 \in \{\blacklozenge, \blacktriangle\}^*$, $\mathcal{I}_M^{\Delta_2 \circ \Delta_1} \subseteq \mathcal{I}_M^{\Delta_1} \cup \mathcal{I}_M^{\Delta_2}$

Proof

For each $\mathfrak{m} \in \uplus M$, $\mathfrak{n} = \Delta_1(M)/\mathtt{Addr}_M(\mathfrak{m})$ and $\mathfrak{k} = \Delta_2(\Delta_1(M))/\mathtt{Addr}_M(\mathfrak{m})$, the following five different cases are conceivable:

1. $\mathfrak{m} \overset{\Delta_1}{\rightsquigarrow} \mathfrak{m} \overset{\Delta_2}{\rightsquigarrow} \mathfrak{m}$

2. $\mathfrak{m} \overset{\Delta_1}{\rightsquigarrow} \mathfrak{n} \overset{\Delta_2}{\rightsquigarrow} \mathfrak{m} \wedge \mathfrak{m} \ncong \mathfrak{n}$

3. $\mathfrak{m} \overset{\Delta_1}{\rightsquigarrow} \mathfrak{n} \overset{\Delta_2}{\rightsquigarrow} \mathfrak{n} \wedge \mathfrak{m} \ncong \mathfrak{n}$

4. $\mathfrak{m} \overset{\Delta_1}{\rightsquigarrow} \mathfrak{m} \overset{\Delta_2}{\rightsquigarrow} \mathfrak{k} \wedge \mathfrak{m} \ncong \mathfrak{k}$

5. $\mathfrak{m} \overset{\Delta_1}{\rightsquigarrow} \mathfrak{n} \overset{\Delta_2}{\rightsquigarrow} \mathfrak{k} \wedge \mathfrak{m} \ncong \mathfrak{n} \wedge \mathfrak{n} \ncong \mathfrak{k} \wedge \mathfrak{m} \ncong \mathfrak{k}$

According to Definition 16, $\mathfrak{m} \in \mathcal{I}_M^{\Delta_2 \circ \Delta_1}$ only for cases 3, 4 and 5 where $\mathfrak{m} \ncong \Delta_2 \circ \Delta_1(M)/\mathtt{Addr}_M(\mathfrak{m})$. For cases 3 and 5, $\mathfrak{m} \in \mathcal{I}_M^{\Delta_1}$, as $\mathfrak{m} \ncong \mathfrak{n}$, thus the theorem is evident. To complete the proof, we ought to demonstrate for case 4 that $\mathfrak{m} \ncong \Delta_2(M)/\mathtt{Addr}_M(\mathfrak{m})$, and therefore is a member of $\mathcal{I}_M^{\Delta_2}$.

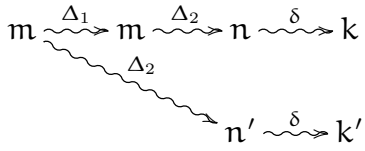
We do this demonstration by structural induction over Δ_2:

The Base Case: For $\Delta_2 = \blacklozenge(\mathfrak{m}, \mathfrak{i}, \nu)$ or $\Delta_2 = \blacktriangle(\mathfrak{m}, \mathfrak{i})$. is already established in Lemma 3.3.2.

Induction Step: We have to show that the theorem holds in case 4 for $\Delta_2' = \delta \circ \Delta_2$ where $\delta \in \{\blacklozenge, \blacktriangle\}$. To simplify the argument, let

$$\begin{aligned} \mathtt{Addr}_M(\mathfrak{m}) &= \mu \\ \mathfrak{n} &= \Delta_2 \circ \Delta_1(M)/\mu \\ \mathfrak{k} &= \delta \circ \Delta_2 \circ \Delta_1(M)/\mu \\ \mathfrak{n}' &= \Delta_2(M)/\mu \\ \mathfrak{k}' &= \delta \circ \Delta_2(M)/\mu \end{aligned}$$

The following diagram illuminates these relationships.



The induction premise is that

$$\mathfrak{n} \ncong \mathfrak{m} \Rightarrow \mathfrak{n}' \ncong \mathfrak{m}$$

We need to demonstrate the induction step, that is:

$$\mathfrak{k} \ncong \mathfrak{m} \Rightarrow \mathfrak{k}' \ncong \mathfrak{m}$$

We prove the induction step by contradiction, that is to say, we assume that $\mathfrak{k}' = \mathfrak{m}$. Similar to the base case, we have the following two possibilities: $\delta = \blacklozenge(\square/\alpha@\mathfrak{a}_i \mapsto \nu)$ or $\delta = \blacktriangle(\square/\alpha.\mathfrak{c}_i)$. From Lemma 3.3.2, we know that in either case $\mathcal{I}^{\delta}_{M} = \{M/\alpha\}$, eitherway. Therefore, the only possible way for $\mathfrak{m} \cong \mathfrak{k}'$ is that $\alpha = \mu$. We discuss each case for δ individually:

case I $\delta = \blacklozenge(\square/\mu@\mathfrak{a}_i \mapsto \nu)$: It follows that $\mathfrak{k}'@\mathfrak{a}_i = \mathfrak{m}@\mathfrak{a}_i = \nu \neq \mathfrak{m}'@\mathfrak{a}_i$. Since $\mathfrak{n} \overset{\delta}{\rightsquigarrow} \mathfrak{k}$, thus $\mathfrak{k}@\mathfrak{a}_i = \nu$, too. Thus,

$$\mathfrak{k} \ncong \mathfrak{m} \Rightarrow \exists\, \mathfrak{a}_j \neq \mathfrak{a}_i.\ \mathfrak{m}@\mathfrak{a}_j \neq \mathfrak{k}@\mathfrak{a}_j = \mathfrak{n}@\mathfrak{a}_j \vee \exists\, \mathfrak{c}_j.\ |\mathfrak{m}/\mathfrak{c}_j| \neq |\mathfrak{k}/\mathfrak{c}_j|$$

For the first case we have:

$$\begin{array}{ll} & \exists\, \mathfrak{a}_j \neq \mathfrak{a}_i.\ \mathfrak{m}@\mathfrak{a}_j \neq \mathfrak{k}@\mathfrak{a}_j \\ \therefore & (\Delta_2 = \Delta_2'; \blacklozenge(M/\mu@\mathfrak{a}_j \mapsto \mathfrak{k}@\mathfrak{a}_j); \Delta_2'') \\ & \wedge(\forall\, \nu.\ \Delta_2'' = \Delta_2'''; \blacklozenge(M/\mu@\mathfrak{a}_j \mapsto \nu); \Delta_2'''' \Rightarrow \nu = \mathfrak{k}@\mathfrak{a}_j) \\ \therefore & \mathfrak{m}'@\mathfrak{a}_j = \mathfrak{k}'@\mathfrak{a}_j = \mathfrak{n}@\mathfrak{a}_j = \mathfrak{k}@\mathfrak{a}_j \end{array}$$

But this contradicts $\mathfrak{m} \cong \mathfrak{k}'$. Similarly, we have:

$$\begin{array}{ll} & \exists\, \mathfrak{c}_j.\ |\mathfrak{m}/\mathfrak{c}_j| \neq |\mathfrak{k}/\mathfrak{c}_j| \\ \therefore & \Delta_2 = \Delta_2'; \blacktriangle(M/\mu/\mathfrak{c}_j); \Delta_2'' \\ \therefore & |\mathfrak{m}'/\mathfrak{c}_j| = |\mathfrak{k}'/\mathfrak{c}_j| \neq |\mathfrak{m}/\mathfrak{c}_j| \end{array}$$

Which is a contradiction.

case II $\delta = \blacktriangle(\square/\mu.\mathfrak{c}_i)$: This implies $|\mathfrak{k}'/\mathfrak{c}_i| > |\mathfrak{m}'/\mathfrak{c}_i|$. But $\mathfrak{k}' \cong \mathfrak{m}$ implies $|\mathfrak{m}/\mathfrak{c}_i| = |\mathfrak{k}'/\mathfrak{c}_i|$ which leads to contradiction, inasmuch as
$\Delta_2, \delta \in \{\blacklozenge, \blacktriangle\}^* \Rightarrow |\mathfrak{m}/\mathfrak{c}_i| \leqslant |\mathfrak{m}'/\mathfrak{c}_i|$.

□

Lemma 3.3.5

$$\mathcal{I}_M^{\blacktriangledown^k(m/i)} = \bigcup_{j=|m/i|-k+1}^{|m/i|} \uplus m/i.j$$

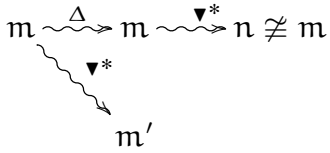
Proof. Straight forward induction using Lemma 3.3.2.

□

Lemma 3.3.6 *If Δ and $\blacktriangledown^*$ are respectively delete-free and delete-only change sequences, then*

$$\mathcal{I}_M^{\blacktriangledown^* \circ \Delta} \subseteq \mathcal{I}_M^{\blacktriangledown^*} \cup \mathcal{I}_M^{\Delta}$$

Proof. Similar to the proof of Theorem 3.3.2, we ought to consider the following and show that $m' \ncong m$.



However,

$$m \ncong n \Rightarrow (n = \Delta(M)/\mu = \bot) \vee (\exists\ c_i\ (|m/c_j| \neq |n/c_i|) \wedge |m/c_i| > 0)$$

For the first case we have:

$$\begin{array}{ll} & n = \Delta(M)/\mu = \bot \\ \therefore & \exists\ \rho, c_i, j.\ m \in \uplus M/\rho/c_i.j \wedge \blacktriangledown^* = \blacktriangledown_1^*; \blacktriangledown(\Box/\rho/c_i); \blacktriangledown_2^* \\ \therefore & m \overset{\blacktriangledown_1^*}{\rightsquigarrow} \overset{\blacktriangledown(\Box/\rho/c_i)}{\rightsquigarrow} \overset{\blacktriangledown_2^*}{\rightsquigarrow} \bot \end{array}$$

And for the second case,

$$\begin{array}{ll} & \exists\ c_i\ (|m/c_j| \neq |n/c_i|) \wedge |m/c_i| > 0 \\ \therefore & \blacktriangledown^* = \blacktriangledown_1^*; \blacktriangledown(\Box/\mu/c_i); \blacktriangledown_2^* \\ \therefore & |m'/c_i| < |m/c_i| \\ \therefore & m \ncong m' \end{array}$$

□

Theorem 3.3.2, Lemma 3.3.5 and Lemma 3.3.6 along with the normalization rules of Section 3.3.1 constitute an algebraic apparatus to calculate an upper-bound for the impact of a composite change on a model, thereby enabling us to perform change impact analysis in a mechanical way, i.e., without actually applying the changes. First, we use the normalization rules to move all the delete operations to the end of the change script and obtain a composite change $\blacktriangledown^* \circ \Delta$, where Δ includes no delete operations. We use Theorem 3.3.2 to calculate $\mathcal{I}^{\Delta}_{\mathsf{M}}$ and Lemma 3.3.5 to calculate $\mathcal{I}^{\blacktriangledown^*}_{\mathsf{M}}$. Finally we combine these two to postulate an upper-bound for the total composite change according to Lemma 3.3.6.

Definition 17 (Reset Operator) *Reset attribute operator, $\blacklozenge^{\perp}$, is a special operator that sets an attribute's value to $\perp$, a reserved value that can only be produced by this operator, and Insertion. The purpose of reset operator is to analyze the intrinsic impact of an update on a model element regardless of the current values that its attributes hold. When applied on an entire model, it resets the values of all attributes.*

For a complex change Δ, the impact set of this change on model M depends on the current values of attributes in the model as well as the atomic change operations that constitute Δ. Some of these change operations, although explicitly define an update value for an attribute, do not actually alter their target attribute value, simply because the current value is the same as the one that the change operator enforces. This prohibits the inclusion of the model element to be accounted for in the impact set. The same element however, only if it assumed a different value for that attribute, would be present in the impact set. For deducing the impact of composite changes from that of their components, it is essential to isolate the effect of the current values of attributes in target model on the impact set. Therefore we define intrinsic impact of change Δ on model M as the set of all elements in M that Δ touches regardless of whether they render differently from their origin or not. This set would be equivalent to the real impact of Δ if all the updates affect their target elements. Therefore, this set is equivalent of the impact of Δ on the reset version of model M, as by definition, it is not possible for any other change to map an attribute's value to $\perp$. Hence the intrinsic impact of Δ on model M is equivalent to $\mathcal{I}^{\Delta}_{\blacklozenge^{\perp}(\mathsf{M})}$. The real impact of a change is a subset of its intrinsic impact : $\mathcal{I}^{\Delta}_{\mathsf{M}} \subseteq \mathcal{I}^{\Delta}_{\blacklozenge^{\perp}(\mathsf{M})}$. The difference of these two sets are the elements that are mapped to their same current values.

3.3.3 Model Transformations

Transformations can be mathematically expressed as mappings between source and target domains. With an analogy to formal languages terminology, a metamodel (or a domain model) can be considered as a generative grammar for that domain. Therefore metamodel $\mathcal{L}(MM_d)$ defines the language of the metamodel MM_d, i.e., the (possibly infinite) set of all models that conform to this metamodel. *Endogenous* transformations, the transformations whose source and target models conform to the same metamodel, as defined in [20] can be expressed as $T_{en} : \mathcal{L}(MM_d) \rightarrow \mathcal{L}(MM_d)$. Similarly, *exogenous* transformations which have different source and target domains can be formulated as $T_{ex} : \mathcal{L}(MM_s) \rightarrow \mathcal{L}(MM_t)$.

Changes in artifacts can be described as endogenous transformations. $\Delta_d : \mathcal{L}(MM_d) \rightarrow \mathcal{L}(MM_d)$ defines the transformation Δ_d over the domain d and applicable to models conforming to metamodel MM_d. Using this notation, interdependent model synchronization becomes the problem of finding the most computationally efficient change operations $M'_t = \Delta_t(M_t)$ applicable to model $M_t : MM_t$ for change $M'_s = \Delta_s(M_s)$ applied on model $M_s : MM_s$ such that M'_t and M'_s remain consistent according to consistency criteria between M_s and M_t.

Artifact generation is, generally, an example of exogenous model transformation as its source and target's metamodels are not necessarily the same. Mapping $T_{Gen} : MM_s \rightarrow MM_t$ transforms models of domain s that conform to metamodel MM_s to models of domain t conforming to metamodel MM_t.

We define consistency between two models in general as a mathematical relation defined over the Cartesian product of their metamodel languages. For example, models $M_1 : MM_1$ and $M_2 : MM_2$ are consistent with respect to consistency relationship
$R \subseteq \mathcal{L}(MM_1) \times \mathcal{L}(MM_2)$ if $(M_1, M_2) \in R$.

3.3.4 Classification of Transformations

Model transformation can be classified into certain groups based on the behavior of the transformation with respect to the change operations applied on the source model. As described, these changes correspond to a set of changes that need to be applied on the target side and result in a consistent model.

Definition 18 (Homomorphic Transformation) *Transformation* T *is said to be homomorphic if* $\mathsf{T}(\Delta(\mathsf{M})) = \Delta'(\mathsf{T}(\mathsf{M})) \Rightarrow \mathsf{T}(\Delta \circ \delta(\mathsf{M})) = \Delta'(\mathsf{T}(\delta(\mathsf{M})))$*, that is to say, the following diagram commutes.*

$$\begin{array}{ccccc} \mathsf{M} & \xrightarrow{\delta} & \delta(\mathsf{M}) & \xrightarrow{\Delta} & \Delta(\delta(\mathsf{M})) \\ & & \downarrow{\scriptstyle \mathsf{T}} & & \downarrow{\scriptstyle \mathsf{T}} \\ & & \mathsf{T}(\delta(\mathsf{M})) & \xrightarrow{\Delta'} & \Delta'(\mathsf{T}(\delta(\mathsf{M}))) \end{array}$$

Definition 19 (Uniform Transformation) *Transformation* T *is said to be uniform if* $\mathsf{T}(\Delta(\mathsf{M})) = \Delta'(\mathsf{T}(\mathsf{M})) \Rightarrow \mathsf{T}(\Delta \circ \Delta(\mathsf{M})) = \Delta' \circ \Delta'(\mathsf{T}(\mathsf{M}))$*, that is to say, the following diagram commutes.*

$$\begin{array}{ccccc} \mathsf{M} & \xrightarrow{\Delta} & \mathsf{M}' & \xrightarrow{\Delta} & \mathsf{M}'' \\ {\scriptstyle \mathsf{T}}\downarrow & & \downarrow{\scriptstyle \mathsf{T}} & & \downarrow{\scriptstyle \mathsf{T}} \\ \mathsf{N} & \xrightarrow{\Delta'} & \mathsf{N}' & \xrightarrow{\Delta'} & \mathsf{N}'' \end{array}$$

Corollary 3.3.7 *A homomorphic transformation is uniform.*

Lemma 3.3.8 *For homomorphic transformation* T *an update always translates to a sequence of updates, i.e.,* $\blacklozenge \xrightarrow{\mathsf{T}} \blacklozenge^*$

Proof. Suppose $\blacklozenge(\square/\alpha, \mathfrak{a}, \nu) \xrightarrow{\mathsf{T}} \delta$. $\mathsf{M}' = \blacklozenge(\mathsf{M}/\alpha@\mathfrak{a} \mapsto \nu)$ is a fixed-point of function $\blacklozenge(\square/\alpha, \mathfrak{a}, \nu)$. Thus from T being a homomorphism it follows that its transformed counterpart, $\delta(\mathsf{T}(\mathsf{M}))$, is also a fixed-point for δ:

$$\begin{array}{ccc} & & \blacklozenge(\mathfrak{a}\mapsto\nu) \\ \mathsf{M} & \xrightarrow{\blacklozenge(\mathfrak{a}\mapsto\nu)} & \mathsf{M}' \\ {\scriptstyle \mathsf{T}}\downarrow & & \downarrow{\scriptstyle \mathsf{T}} \\ \mathsf{N} & \xrightarrow{\delta} & \mathsf{N}' \\ & & \delta \end{array}$$

To have a fixed-point, δ cannot have any $\blacktriangle$ operations in its normalized form. Suppose $\delta = \blacktriangledown(\square/\beta) \circ \delta'$. For N' to be a fixed-point for δ it requires that $\mathsf{N}'/\beta = \varnothing$. It follows that the container pointed to by β should also be empty in $\mathsf{N} = \mathsf{T}(\mathsf{M})$, as it is also equal

to $\mathsf{T}(\blacklozenge(M'/\alpha@\mathfrak{a} \mapsto M/\alpha@\mathfrak{a}))$, i.e., $N/\beta = \varnothing$, otherwise we would need to have for the reverse update operation: $\blacklozenge(\Box/\alpha@\mathfrak{a} \mapsto M/\alpha@\mathfrak{a}) \xrightarrow{\mathsf{T}} \blacktriangle(\Box/\beta) \circ \delta''$, which was just proved to be impossible. Therefore, the normalized form of δ does not comprise any deletion either. Thus $\delta = \blacklozenge^*$.

□

Lemma 3.3.9 *For homomorphic transformation* T

$$\blacktriangle(\Box/\alpha) \xrightarrow{\mathsf{T}} \blacklozenge(\Box/\beta@\mathfrak{a} \mapsto \nu) \Rightarrow \forall M \quad \mathsf{T}(M)/\beta@\mathfrak{a} = \nu$$

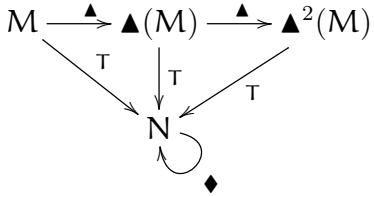
Proof. Suppose $\blacktriangledown(\Box/\alpha) \xrightarrow{\mathsf{T}} \delta$, for any M we have

$$\mathsf{T}(M) = \mathsf{T}(\blacktriangledown(\blacktriangle(M/\alpha))) = \delta(\mathsf{T}(\blacktriangle(M)) = \delta(\blacklozenge(\mathsf{T}(M)/\beta@\mathfrak{a} \mapsto \nu))$$

if $\mathsf{T}(M)/\beta@\mathfrak{a} = \chi \neq \nu$ then we should have $\delta = \blacklozenge(\Box/\beta@\mathfrak{a} \mapsto \chi)$, but this results in contradiction:

$$\begin{aligned} \mathsf{T}(\blacktriangle^2(M/\alpha)) &= \blacklozenge^2(\mathsf{T}(M)/\beta@\mathfrak{a} \mapsto \nu) \\ &= \blacklozenge(\mathsf{T}(M)/\beta@\mathfrak{a} \mapsto \nu) \\ \mathsf{T}(\blacktriangledown(\blacktriangle^2(\mathsf{T}(M)/\alpha))) &= \mathsf{T}(\blacktriangle(M/\alpha)) \\ &= \blacklozenge(\mathsf{T}(M)/\beta@\mathfrak{a} \mapsto \nu) \end{aligned}$$

or diagrammatically:



but also:

$$\begin{aligned} \mathsf{T}(\blacktriangledown(\blacktriangle^2(M/\alpha))) &= \blacklozenge(\mathsf{T}(\blacktriangle^2(M/\alpha))/\beta@\mathfrak{a} \mapsto \chi) \\ &= \blacklozenge(\blacklozenge(\mathsf{T}(M)/\beta@\mathfrak{a} \mapsto \nu)/\beta@\mathfrak{a} \mapsto \chi) \\ &= \mathsf{T}(M) \end{aligned}$$

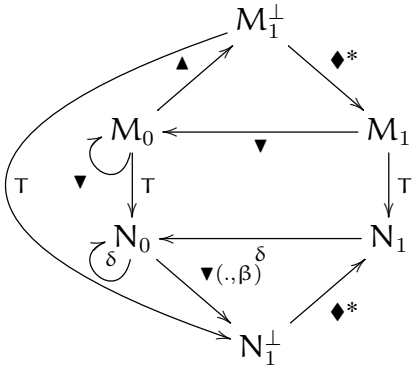
□

Lemma 3.3.10 *For homomorphic transformation* T

$$\blacktriangle(\Box/\alpha) \xrightarrow{\mathsf{T}} \blacktriangledown(\Box/\beta) \Rightarrow \forall \mathsf{M} \;\; \mathsf{T}(\mathsf{M})/\beta = \varnothing$$

Proof. Suppose M is a model element, then $\mathsf{M}_0 = \blacktriangledown^{|\mathsf{M}/\alpha|}(\mathsf{M}/\alpha)$ is a fixed point for function $\blacktriangledown(\Box/\alpha)$. If $\blacktriangledown(\Box/\alpha) \xrightarrow{\mathsf{T}} \delta$ the following set of equations holds for $\mathsf{N}_0 = \mathsf{T}(\mathsf{M}_0)$ and $\mathsf{N}_1^{\perp} = \mathsf{T}(\mathsf{M}_1^{\perp}) = \mathsf{T}(\blacktriangle(\mathsf{M}_0/\alpha))$.

$$\begin{aligned} \blacktriangledown(\mathsf{N}_0/\beta) &= \mathsf{N}_1^{\perp} \\ \delta(\mathsf{N}_1^{\perp}) &= \mathsf{N}_0 \\ \delta(\mathsf{N}_0) &= \mathsf{N}_0 \end{aligned}$$



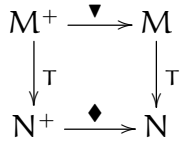
The third equation is the translation of the fact that M_0 is a fixed-point of $\blacktriangledown(\Box/\alpha)$ to the other side of the transformation. These equations only have one solution: $\mathsf{N}_0 = \mathsf{N}_1^{\perp}$, $\mathsf{N}_0/\beta = \varnothing$ and $\delta = \mathtt{identity}$. From Lemma 3.3.8 it follows that $\mathsf{N}_1/\beta = \varnothing$. A straight forward induction using the same argument yields that $\mathsf{N}/\beta = \varnothing$.

□

Lemma 3.3.11 *For homomorphic transformation* T

$$\blacktriangledown(\Box/\alpha) \xrightarrow{\mathsf{T}} \blacklozenge(\Box/\beta@\mathfrak{a} \mapsto \nu) \Rightarrow \forall\ \mathsf{M}\ \mathsf{T}(\mathsf{M})/\beta@\mathfrak{a} = \nu$$

Proof. For model M let $\mathsf{N} = \mathsf{T}(\mathsf{M})$, $\mathsf{M}^{+} = \blacktriangle(\mathsf{M}/\alpha)$, $\mathsf{N}^{+} = \mathsf{T}(\mathsf{M}^{+})$.



As the above diagram shows $N = \blacklozenge(N^+/\beta@a \mapsto v)$ thus we can infer that $N/\beta@a = v$.

□

Lemma 3.3.12 *The normalized translation of a delete operation by a homomorphic transformation* T *cannot comprise any insertion.*

Proof. For the change $\blacktriangledown(\square/\alpha)$ consider M_0 to be its fixed-point, i.e., $|M_0/\alpha| = 0$, and let $N_0 = T(M_0)$. Assume that there is an un-cancelled insert in the mapped change sequence.

$$\begin{aligned} & T(\blacktriangledown(M_0/\alpha)) = N' = \delta' \circ \blacktriangle(\square/\beta) \circ \delta(N_0) \\ \therefore \quad & |N'/\beta| \geqslant |N_0/\beta| + 1 \end{aligned}$$

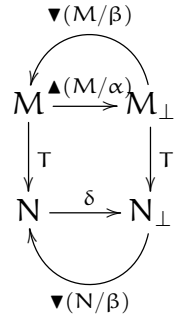
Which is a contradiction, since $N' = T(M_0) = N_0$.

□

Lemma 3.3.13 *For homomorphic transformation* T

$$\blacktriangledown(\square/\alpha) \xrightarrow{T} \blacktriangledown(\square/\beta) \Rightarrow \blacktriangle(\square/\alpha) \xrightarrow{T} \blacklozenge^* \circ \blacktriangle(\square/\beta)$$

Proof. From the following diagram and edit sequence simplification rules,



it is evident that $\delta = \blacklozenge^* \circ \blacktriangle(N/\beta)$.

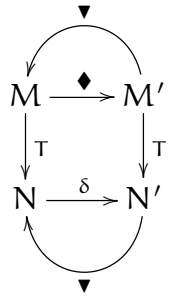
□

Lemma 3.3.14 (Locality) *For homomorphic transformation* T

$$\blacktriangledown(\square/\alpha) \xrightarrow{\mathsf{T}} \blacktriangledown(\square/\beta) \wedge \blacklozenge(\square/\alpha.\mathfrak{i}@\mathfrak{a} \mapsto \nu) \xrightarrow{\mathsf{T}} \blacklozenge(\square/\gamma.\mathfrak{j}@\mathfrak{a}' \mapsto \nu') \Rightarrow \begin{cases} \gamma = \beta.\mathfrak{i}/\mathfrak{p} & \text{or} \\ \mathsf{T}(\mathsf{M})/\gamma.\mathfrak{j}@\mathfrak{a}' = \nu & \forall\ \mathsf{M} \end{cases}$$

For some attribute $\mathfrak{a}'$ *and value* ν'*.*

Proof. For model M first let $\mathfrak{i} = |\mathsf{M}/\alpha|$. In the following diagram δ, since T is homomorphic, is an update operation.



Let $\delta = \blacklozenge(\square/\gamma.\mathfrak{j}@\mathfrak{a}' \mapsto \nu')$. If $\mathsf{N}'/\gamma \not\subseteq \uplus\mathsf{N}'/\beta.\mathfrak{i}$–that is to say, if the element at address $\beta.\mathfrak{i}$ is not a parent of the elements in container γ–then $\blacktriangledown(\mathsf{N}'/\beta)/\gamma.\mathfrak{j}@\mathfrak{a}' = \nu'$, because the element at $\gamma.\mathfrak{j}$ is not eliminated by the delete operation. But we also have $\mathsf{N} = \blacktriangledown(\mathsf{N}'/\beta)$, therefore, $\mathsf{N}/\gamma.\mathfrak{j}@\mathfrak{a}' = \nu'$. A straight forward induction on $\mathfrak{n} = 1..|\mathsf{M}/\alpha|$ establishes the lemma for any element at $\mathsf{M}/\alpha.(|\mathsf{M}/\alpha| - \mathfrak{n} + 1)$.

□

The above Lemma is important because it tells us that for homomorphic transformations the remote impact of an update is locally bound to the elements contained by the corresponding element on the target side.

Definition 20 (Monotonic Transformation) *Transformation* T *is said to be monotonic if*

$$\begin{aligned} \mathsf{T}(\blacklozenge(\mathsf{M})) &= \blacklozenge^*(\mathsf{T}(\mathsf{M})) \\ \mathsf{T}(\blacktriangle(\mathsf{M})) &= \blacktriangle^*(\mathsf{T}(\mathsf{M})) \\ \mathsf{T}(\blacktriangledown(\mathsf{M})) &= \blacktriangledown^*(\mathsf{T}(\mathsf{M})) \end{aligned}$$

3.3.5 Traces and Dependencies across Transformations

A model element is said to be dependent on (or traced to) another model element through a transformation if making changes to the source element would require making a change to the target element. This notion is defined formally in the following.

Definition 21 (Dependency) *Model elements* $m \in \uplus M$ *and* $n \in \uplus N$ *are said to be dependent through transformation* $N = T(M)$ *if there exist changes* δ *and* δ' *such that* $T(\delta(M)) = \delta'(N) \wedge \mathcal{I}^{\delta}_{M} = \{m\} \wedge n \in \mathcal{I}^{\delta'}_{N}$.

The most common form of dependency between model elements on the target and source side of a transformation is when some attribute values of the element on the target side de depend on some of the attribute values of the element in the source model. The following definition gives a formal account of this type of dependency between model elements.

Definition 22 (Value Dependency) *Model elements* $m \in \uplus M$ *and* $n \in \uplus N$ *are value-dependent if*

$$\exists\ a_i, a_j \quad \blacklozenge(m@a_i) \xrightarrow{T} \blacklozenge(n@a_j)$$

The source and target models are called existentially dependent if removing the former from the source model would entail the removal of the latter from the target model.

Definition 23 (Existential Dependency) *Model elements* $m \in \uplus M$ *and* $n \in \uplus N$ *are existentially dependent through transformation* T *if*

$$\blacktriangledown(M, m) \xrightarrow{T} \blacktriangledown(N, n)$$

Definition 24 *A dependency relationship between two model elements is called purely existential if the two elements are existentially-dependent but they are not value-dependent.*

According to Lemma 3.3.13 if two elements are existentially dependent through a homomorphic transformation then the dependency also extends to their containers. That is, insertion to the source's container also entails insertion to the corresponding target's container. In other words, all elements in the containers of two existentially dependent elements are existentially dependent.

3.3.6 Semantics of Synchronization for Generated Artifacts

Trivially, when the source artifact changes, the same transformation whereby the target artifact was generated from the source artifact can be reinvoked to generate a new version of the target model that is consistent with the modified source. Whether this regeneration of the target artifact is computationally efficient depends on the size of the models and the complexity of the invoked transformation. Nevertheless, the consistency relationship between the source and target models is implicitly enforced by the embedded transformation rules. Thus the problem of model synchronization for the models bound together with transformation rules such as, $\mathsf{M_t} = \mathsf{T_{Gen}}(\mathsf{M_s})$, reduces to determining $\Delta_\mathsf{t} : \mathsf{MM_t} \rightarrow \mathsf{MM_s}$ for every change Δ_s defined on the source model, such that $\Delta_\mathsf{t}(\mathsf{M_t}) = \mathsf{T_{Gen}}(\Delta_\mathsf{s}(\mathsf{M_s}))$. It should be noted that this process does not necessarily involve complete regeneration of the target model. In fact, to have an efficient model synchronizer we would like to translate the changes of the source domain to the minimum set of modifications in the target domain that change the target producing the same model that the transformation would yield if applied on the updated source model.

3.3.7 Bi-directional synchronization

Unlike compilation of programs into machine code, in MDD it cannot be assumed that the end product of a transformation will not be independently changed. Although a model generated from another model is a live software document that independently evolves, the source artifact still has to be reconciled with it accordingly. The model transformer used for generating the target model is not necessarily a bi-directional transformation, hence, not usable for backward synchronization. Although, it may not always be possible to reconcile models in both directions, a supervised synchronization scheme that is capable of propagating changes in both directions can be of tremendous practical value even if it does not provide full consistency. This problem can be stated as finding change function $\Delta_\mathsf{s} : \mathsf{M_s} \rightarrow \mathsf{M_s}$ for a change Δ_t over domain t such that if the transformation $\mathsf{T_{Gen}}$ is re-executed on $\mathsf{M'_s} = \Delta_\mathsf{s}(\mathsf{M_s})$ it would result in $\mathsf{M'_t} = \Delta_\mathsf{t}(\mathsf{M_t})$.

3.3.8 Incremental synchronization

Definition 25 (Incrementality) *For two models* M_s *and* M_t *that are to be synchronized under consistency relationship* $\mathcal{R}$*, synchronizer.* $\mathsf{Sync} : \mathsf{MM}_s \times \mathsf{MM}_s \times \mathsf{MM}_t \to \mathsf{MM}_t \times \mathsf{MM}_t$ *computes the change* $\Delta_t = \mathsf{Sync}_{s,t}(\mathsf{M}_s, \Delta_s(\mathsf{M}_s), \mathsf{M}_t)$ *applicable to model* M_t*.* Sync *is called (Full)-Incremental if the following conditions hold:*

$$\mathcal{I}^{\Delta_s}_{\mathsf{M}_s} = \varnothing \Rightarrow \mathcal{I}^{\Delta_t}_{\mathsf{M}_t} = \varnothing$$
$$\mathcal{I}^{\Delta_t}_{\mathsf{M}_t} = \mathcal{I}^{\Delta_t}_{\blacklozenge^{\perp}(\mathsf{M}_t)}$$
$$\forall \Delta'_t \in \mathsf{MM}_t \times \mathsf{MM}_t \quad (\Delta_s(\mathsf{M}_s), \Delta'_t(\mathsf{M}_t)) \in \mathcal{R} \Rightarrow \mathcal{I}^{\Delta_t}_{\mathsf{M}_t} \subseteq \mathcal{I}^{\Delta'_t}_{\blacklozenge^{\perp}(\mathsf{M}_t)}$$

Definition 26 (Partial Incrementality) Sync *is said to be Partially-Incremental if* $\mathcal{I}^{\Delta_t}_{\mathsf{M}_t} \subset \uplus \mathsf{M}_t$*.*

Definition 27 (Non-Incrementality) Sync *is called Non-Incremental if* $\mathcal{I}^{\Delta_t}_{\mathsf{M}_t} = \uplus \mathsf{M}_t$*.*

The above definitions utilize the notion of the impact set to give a precise definition for incremental synchronization that does not directly rely on time complexity. The essence of Definition 25 is that it considers a synchronization operation incremental, if for a given change applied to the source model of the transformation, the impact of the change sequence it produces for the target side is smaller than any other change sequence that converts the target model to the new version. In other words, it only alters elements that need to be modified. A non-incremental synchronization always touches all the elements of the target model regardless of the input change. A partially incremental synchronization is in between; its impact is a subset of the target model.

Chapter 4

Incremental Synchronization of Black-box Transformations

In this chapter, we turn our attention to the problem of model synchronization for the situations where the consistency requirements between dependent software artifacts are established by software generators, i.e., when some artifacts are generated by transforming some others using a number of transformations. In this context, there exists a definitive measure for consistency between the source and the target of the generator. If the source changes, it is possible to attain a consistent target by re-applying the generator on the new version of the source artifact. Regeneration, however, can be inefficient if it needs to be done recurrently. An incremental synchronization scheme is one that can reconcile the source with the target of the artifact by only modifying the affected elements in the target, thereby avoiding the superfluous re-computation of unaffected target elements.

The general strategy for deriving incremental synchronization for a given transformation has been to specify the transformation in a framework that supports the execution of transformations in an incremental fashion. This approach has some practical burdens, though: it requires re-implementing an existing piece of software in a new language; a notoriously challenging problem for practitioners. In this chapter, we treat existing transformations as black-boxes and try to build synchronization as an added feature that re-uses the transformation's implementation. This saves the effort required to reverse-engineer the logic of the transformation from the existing implementation and re-implement them in another notation.

4.1 Architecture Overview

The architecture of our proposed solution is depicted in the block-diagram diagram of Figure 4.1. The crux of our idea is to sift the information for all inter-related models into small pieces, store them in one centralized place, and refer to them by a unique identifier across all heterogeneous models. To that end, we propose a process called *Conceptualization*. This process involves identifying, abstracting, tagging and centralizing the data encapsulated within a software artifact into logical entities called *Concepts*. Mutual information in related artifacts can be traced to each other through concepts;Related artifacts share mutual information that can be linked by concepts; two or more inter-related elements which reside in different artifacts may represent the same piece of information by referring to the same concept.

The overall architecture of the black-box synchronization framework is presented in Figure 4.1. Concepts are stored in a centralized location referred to as the *concept pool*. The framework provides facilities to efficiently trace a concept from any given position inside a model, to its corresponding entity in the concept pool and *vice versa*. The synchronizer unit listens for changes in the interrelated artifacts. When a change occurs, the system updates the values of concepts corresponding to the modified elements in the concept pool. The synchronization of inter-dependent models may be conducted lazily, that is, when the models need to be re-synchronized, the values in the affected concepts in the concept pool are propagated to the model elements that are indexed by the same unique identifier of the modified element. This propagation takes the form of value replacement, and can be carried out in linear time with respect to the number of elements involved. As the figure indicates, the synchronizer is, in principle, able to propagate changes in both directions, notwithstanding the subject transformation's support for bi-directionality or lack thereof.

Automatic transformations are used in many software development environments to generate new artifacts from the existing ones. SOAP-baed Web Services is one such domain that incorporates various software specifications. At the very core of a web-service lies the service implementations code, authored in a programming language such as Java. There is a service description denoted in an XML based format called Web Services Description Language (WSDL), which specifies the interface of a web-service. State of the art SOA development tools provide automatic (or semi-automatic) means for the generation of, among other artifacts, WSDL from Java. In Eclipse Web Tools Platform (WTP),

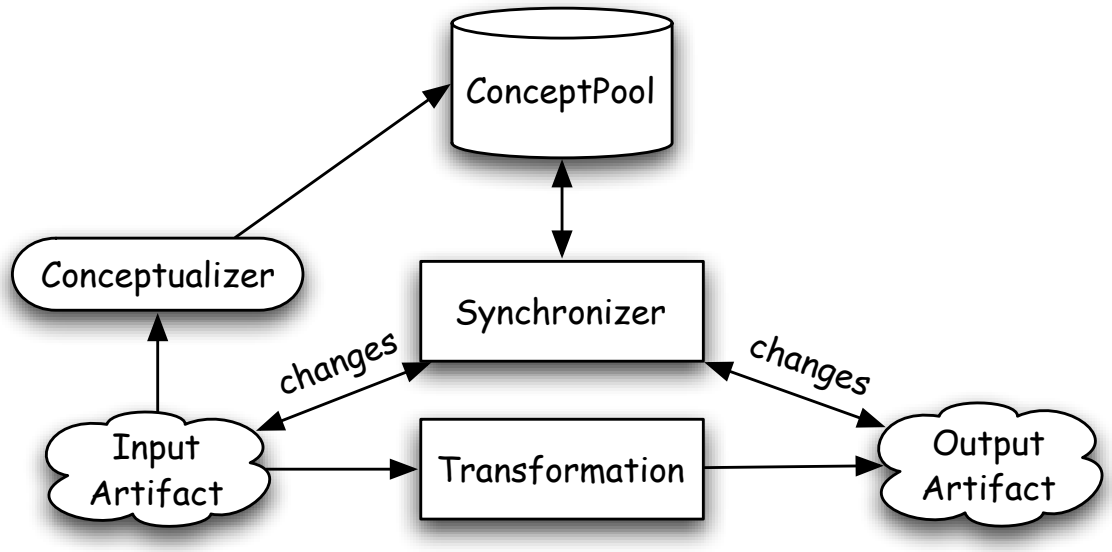


Figure 4.1: Architecture of the Synchronization Framework (arrows denote dataflow)

for example, WSDL can be generated from Java source code via a transformation called Java2WSDL, as depicted in Figure 4.2. When an element of the source artifact, such as the name of the method in this example, is updated, or a method is added to the source code, the target file, e.g., WSDL in this case, has to be changed accordingly. We will use this simplified version of Java2WSDL transformation and its pertaining source and target artifacts of Figure 4.2 as our running example to demonstrate the various steps of our generic incremental model synchronization methodology.

In the running example, the source and target artifacts respectively adhere to the Java grammar and the WSDL schema. To cope with diversity, all artifacts are represented in a canonical representation format; abstraction models, which are defined using a unified meta-metamodel (e.g., MOF or EMF). Figure 4.3 depicts the JavaSrvImpl metamodel for abstracting Java implementations of web services. For brevity, this abstraction only retains the elements pertinent to the Java2WSDL transformation. Thus, the code inside method bodies is filtered out.

Figure 4.4 presents the metamodel for the Web Service Description Language. The top level element is Definition. Each definition contains a number of Service instances, which

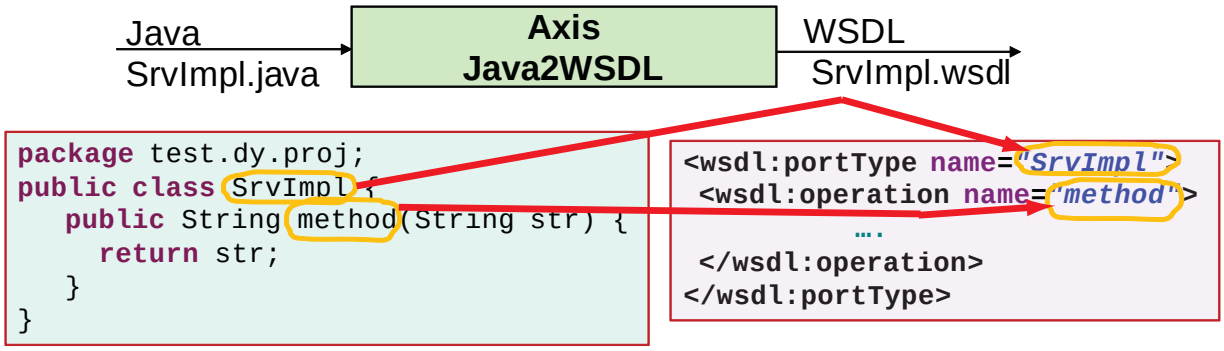


Figure 4.2: Generating WSDL from Java Classes

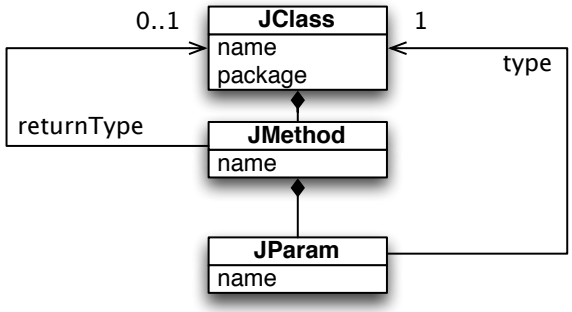


Figure 4.3: Simplified metamodel for Java Implementation of a Web Service

bind to a concrete location by a Binding. Services respond to messages each represented by an instance of the Message type, which also belongs to the top-level Definition type. The concrete interface of each Message is specified by a PortType. The WSDL metamodel has some peculiarities, too. Specifically, WSDL features a type definition section which corresponds to XML schema metamodel. In other words, the elements defined in the type definition section of WSDL, ExtensibilityElements, are subclasses of XMLSchema. Figure 4.5 illustrates the abstracted models of Figure 4.2 according to these two metamodels.

4.1.1 Overall Process

When a software artifact is changed, the first synchronization step is to reconcile the rest of the elements in the same artifact with the changed ones. We refer to this step as

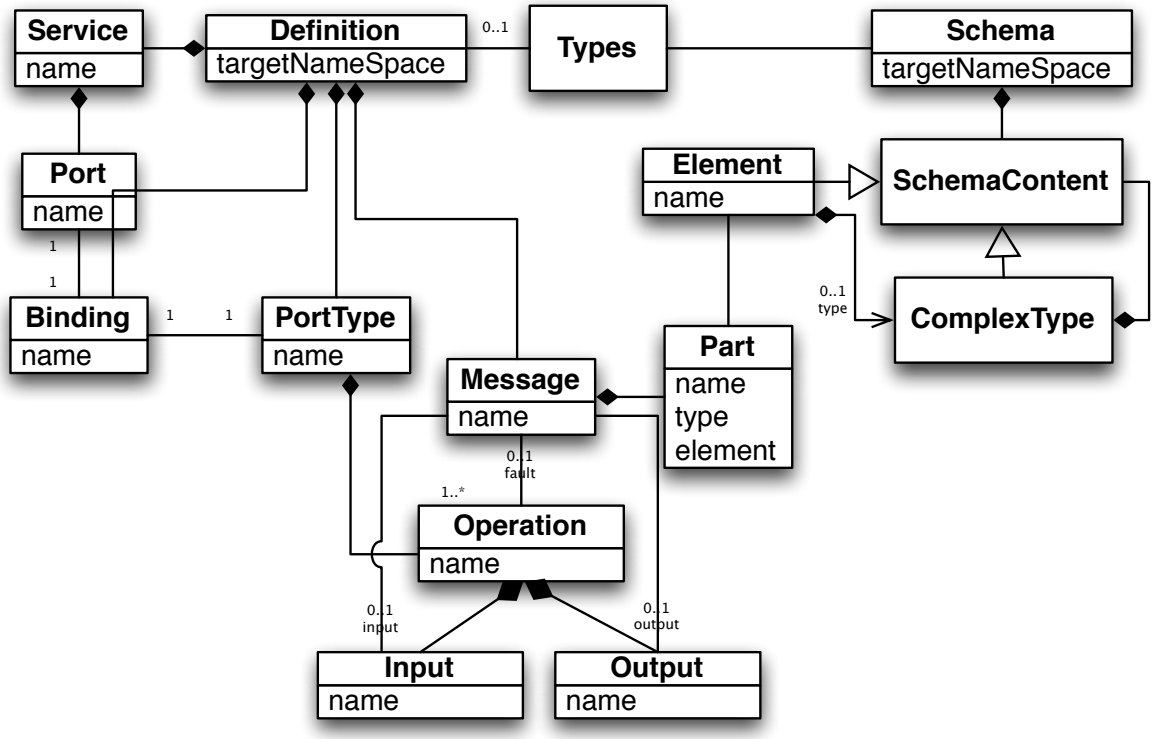


Figure 4.4: Simplified metamodel of WSDL 2.0

Intra-Model Change Propagation, for its impact is limited to the boundary of the changed artifact. The second step, *Inter-Model Change Propagation*, propagates the changes made to a software artifact to other inter-dependent artifacts in the system.

To have a fully synchronized model of the subject system, it is necessary to carry out both types of propagation. When a change is induced to an artifact, it should first be propagated inside the same artifact by triggering the Intra-Model Change Propagation strategies. Having made the artifact coherent within itself, the next phase is to synchronize other interrelated artifacts with the altered one via Inter-Model Change Propagation. Yet the process does not end at this step; the changes made to other artifacts can trigger additional inconsistencies between models and even to the original changed model. Our solution based on centralized concept pool does not need repetitive synchronization performed in such cases. In contrast, it is capable of resolving inconsistencies of concepts

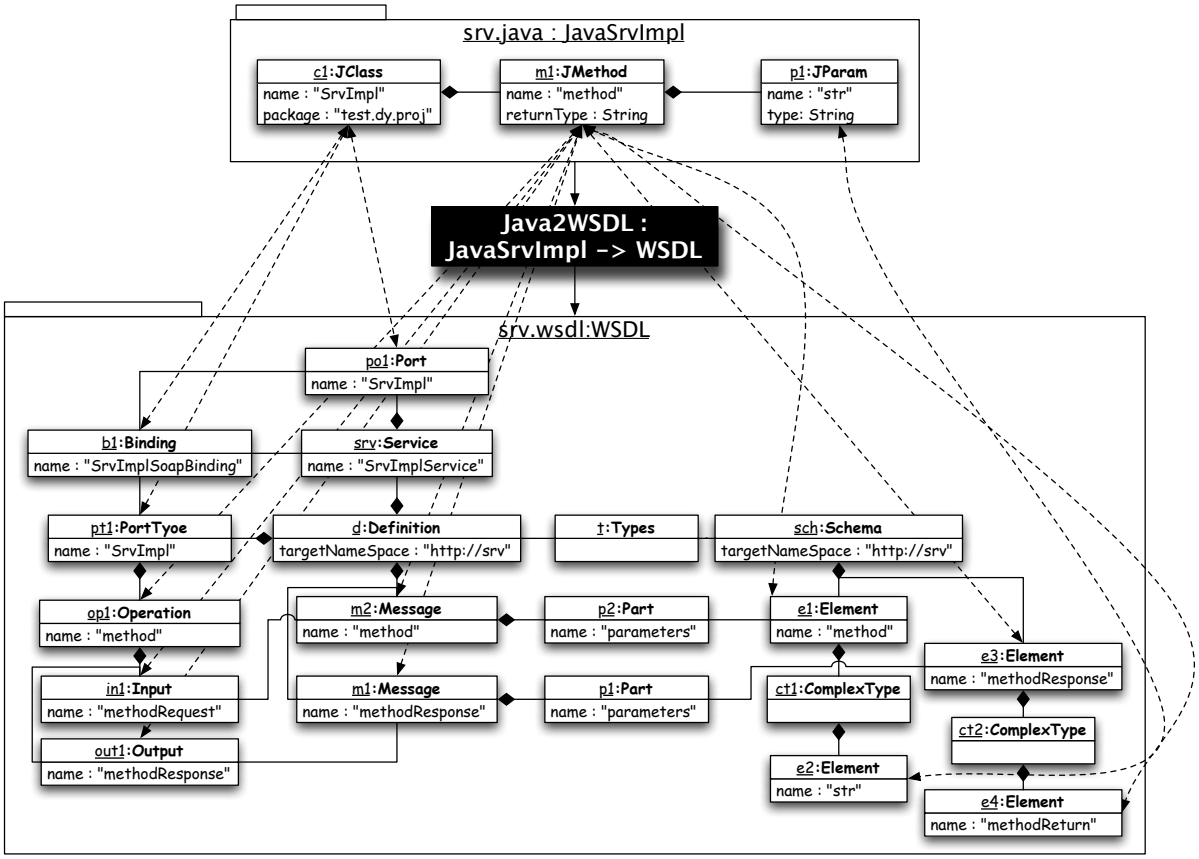


Figure 4.5: Abstract Models of Java and WSDL

residing in multiple models in one shot, due to the fact that these artifacts all refer to the centralized concept pool for fetching new values. The general process for carrying out change propagation in a multi-model environment is described in the following.

Synchronization Process

```
1  Check preconditions and Control Strategy for termination
2  Perform Intra Model Change Propagation for changed artifacts
3  Check Post Conditions
4  Calculate list of all affected artifacts
5  For each artifact in the impact list
6        Perform Inter-Model Update Propagation
7  Verify the integrity of altered models
```

As we shall discuss later, multiple consistency requirements can give rise to non-terminating chains of synchronization, which should, in general, be supervised by a control strategy to break non-terminating cycles. The pre and post conditions verify that each step of the process results in well-formed artifacts.

4.1.2 Conceptualization Phase

Concepts are defined as primitive abstract entities that semantically associate two or more information carrying elements across a pool of heterogeneous software models. Models consist of model elements, which enclose several attributes to represent information. A concept, however, can be even smaller than an attribute value; attribute values can be composed of multiple concepts. Concepts, ideally, represent quanta of information, which cannot be broken down into smaller pieces. From this point of view, models provide organization, structure and semantics to an amalgamation of concepts by encapsulating them into various model elements of different types.

Furthermore, related artifacts share mutual information. Conceptualization assists the synchronization of this mutual information in two major ways. First, concepts provide a systematic way for tracing piecemeal the propagation of transformed data, inside and outside the boundaries of the artifact in which they are located. Second, concepts can establish fine-grained interdependencies between two or more inter-related artifacts; different elements in multiple artifacts can be made represent the same concept by referencing its unique identifier. *Conceptualization* is the process of extracting, indexing, tagging and

centralizing concepts. Concepts are stored in a database embedded in the development environment. This database is referred to as the *Concept Pool*.

Conceptualization can directly resolve intra-model and intter-model inconsistencies. Conceptualized artifacts have the property that their interdependent elements share mutual information through centralized concepts. Therefore, a change to one of the elements causes the related concept values be updated in the concept pool. An update can, thus, be easily propagated by pushing the values of the modified concepts to all referencing elements. Inter-model change propagation can also be embodied by utilizing the conceptualization process, granted that the models' dependent elements are linked by referring to the same concepts in the concept pool.

Definition 28 (Concept) *A concept* $\mathfrak{c} \in \mathcal{ID} \times \Sigma^* \times 2^{\mathbf{Addr}}$ *is a tuple;* $\mathcal{ID}$ *is the set of all IDs,* Σ^* *is the set of all values, and* $2^{\mathbf{Addr}}$ *is the powerset of all anonymous addresses.*

We denote $\mathfrak{a} \hookrightarrow \mathfrak{c} \in \mathcal{CP}$, to state that attribute $\mathfrak{a}$ refers to concept $\mathfrak{c}$ in concept pool $\mathcal{CP}$.

Figure 4.6 illustrates the results of conceptualization for the case of the Java2WSDL example. The dependencies between the source and the target of the transformation are established using concepts, as illustrated in Figure 4.6.

Propagating changes from one model to another in this scheme takes the form of updating the pertinent concepts in the concept pool followed by fetching new values to each affected element, provided that there exists a mechanism whereby the system can pinpoint the related concepts in the concept pool for a given model element. For example, in Figure 4.6, if the method argument name *"str"* is changed in the Java model, the framework updates its related concept in the pool and notifies its dependent element in the WSDL side, i.e. WSDL message, to updates its *"name"* attribute with the new value. A modification made to the elements of the target side can likewise be propagated to the source side.

4.1.3 Shadow Phase

As mentioned earlier, it is essential to locate the concepts associated with each model element, and also respond to queries about elements sharing the same concepts. To enable

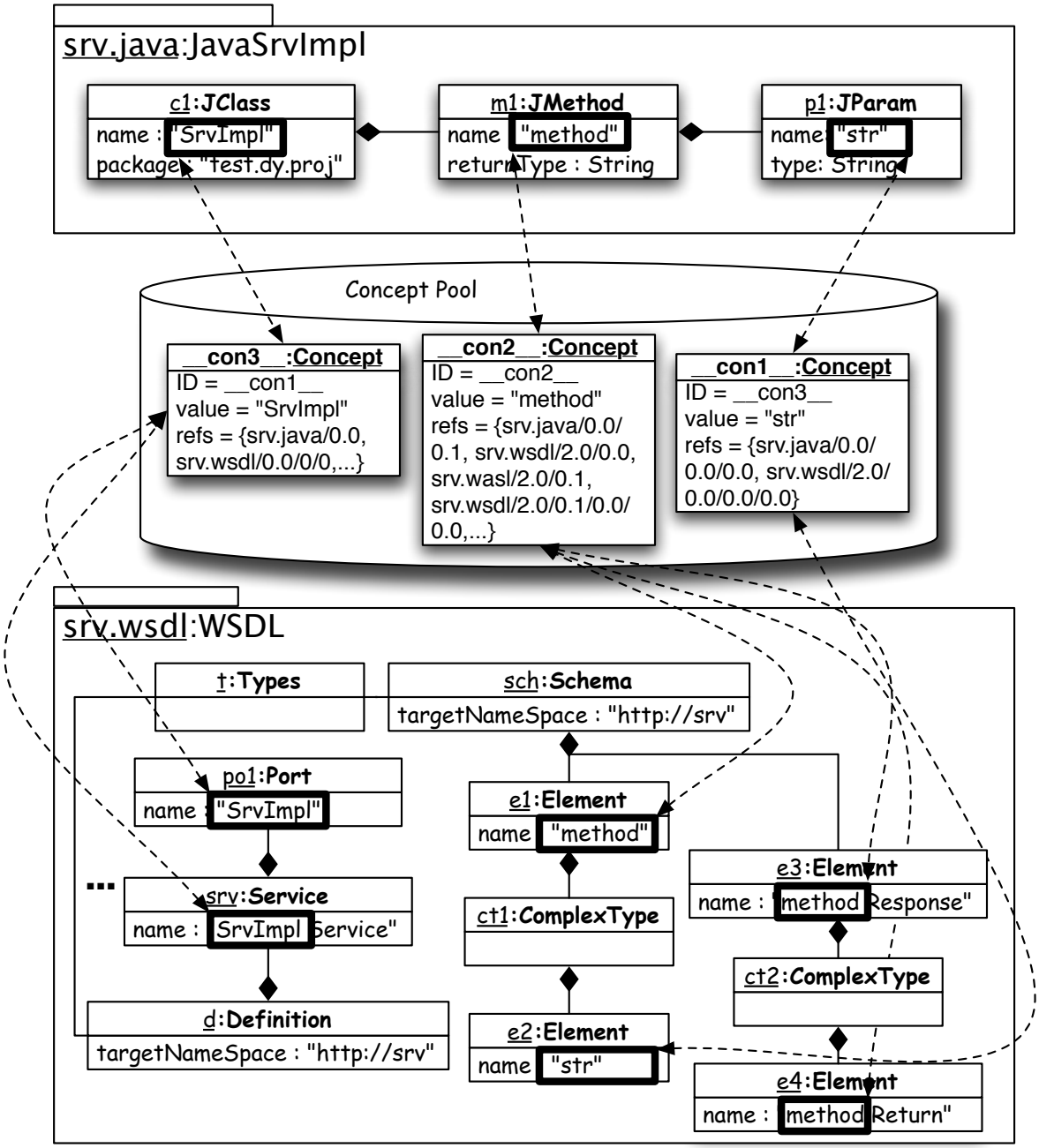


Figure 4.6: Conceptualization of Java and WSDL models

the latter, an index of addresses of related model elements for each concept is stored in

its corresponding entry in the concept pool. To address the former, we propose *Shadow Models*; they are intermediate models intended to facilitate answering to queries about elements sharing the same concepts. Shadow models closely mimic the structure of the original models. They are, in fact, produced by exchanging the values of the identified concepts in the models by their unique identifiers. Shadow models make accessible the identified concepts in the concept pool since a concept entry of an attribute value in the concept pool can be located by obtaining its concept ID from the exact same position of the attribute in the shadow model.

Algorithm 3 Shadow(M)

Input Model $\mathsf{M} = (\mathfrak{C}, \mathsf{A}, \mathfrak{R}, \mathsf{T})$ and Concept-Pool $\mathcal{CP}$
Output Shadow Model S

```
1: function SHADOW(M,CP)
2:     S ← Clone(M)                          ▷ Create a clone of the original model
3:     for all m ∈ S do
4:         for all a_i ∈ A(m) do   ▷ Replace all attributes' values with their concept IDs
5:             ♦(m, i, CP.getConceptID(a_i))
6:         end for
7:     end for
8: return S
9: end function
```

The algorithm for creating shadow models is listed as Algorithm 3. In the algorithm, the value of all conceptualized attributes are replaced by the identifier of their pertinent concepts. An important practical note when generating unique concept identifiers is to ensure them to be valid identifiers with respect to the grammar of both the source and the target artifacts. For the case of Java and WSDL, this means that they need to be constructed using the characters allowed in the Java grammar and the WSDL schema for class and method names and also WSDL identifiers. This requirement is to guarantee that the shadow model is a well-formed artifact of the same type of the original one, and can be used as input to transformers applicable on the original artifact. We insist that shadow artifacts be valid documents of the same type of the source model, because we use them as inputs to the transformation to generate shadow models of the target domain, thereby achieving traceability through the common concepts appearing in the shadow models of both sides of the transformation.

Figure 4.7 depicts the application of the shadow algorithm (Algorithm 3) on the ex-

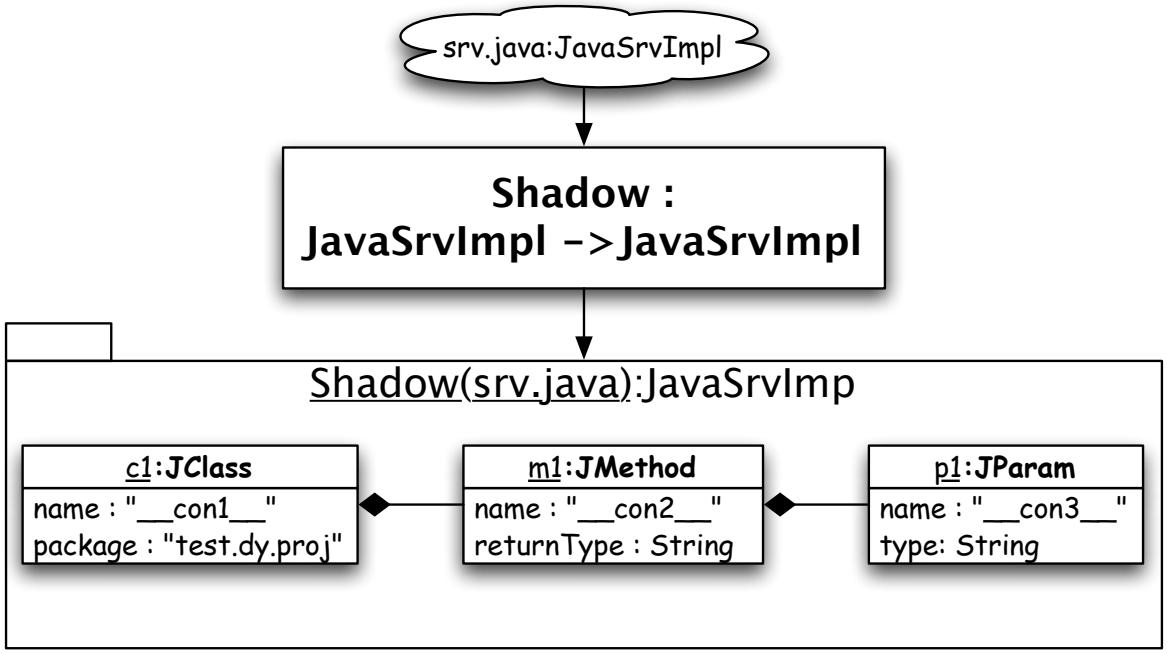


Figure 4.7: Creating Shadow for Java model

ample input model, srv.java.JavaSrvImpl. The output of the shadow function, as shown in the figure, is structurally identical to the original model of Figure 4.5, but the values of its conceptualized attributes are replaced by the unique identifiers of their corresponding concepts. The structural reciprocity between models and their shadows enable us to easily trace each attribute to its related concept(s) in the concept pool; we only ought to look at the exact position of said attribute's element in the shadow model to obtain its concept identifier, and, thereafter, perform a concept lookup in the concept pool.

The shadow of the target model is attained by applying the transformation on the shadow model of the source. Figure 4.8 depicts the application of the Java2WSDL transformation on the shadow of the source Java model, whence the shadow WSDL ensues. The reason for obtaining the target WSDL model via the application of the transformation, rather than using the shadow function on the target model, is to ensure both shadows use the same concept IDs to refer to interlinked elements, as is the case for the Java and WSDL shadows in Figures 4.7 and 4.8.

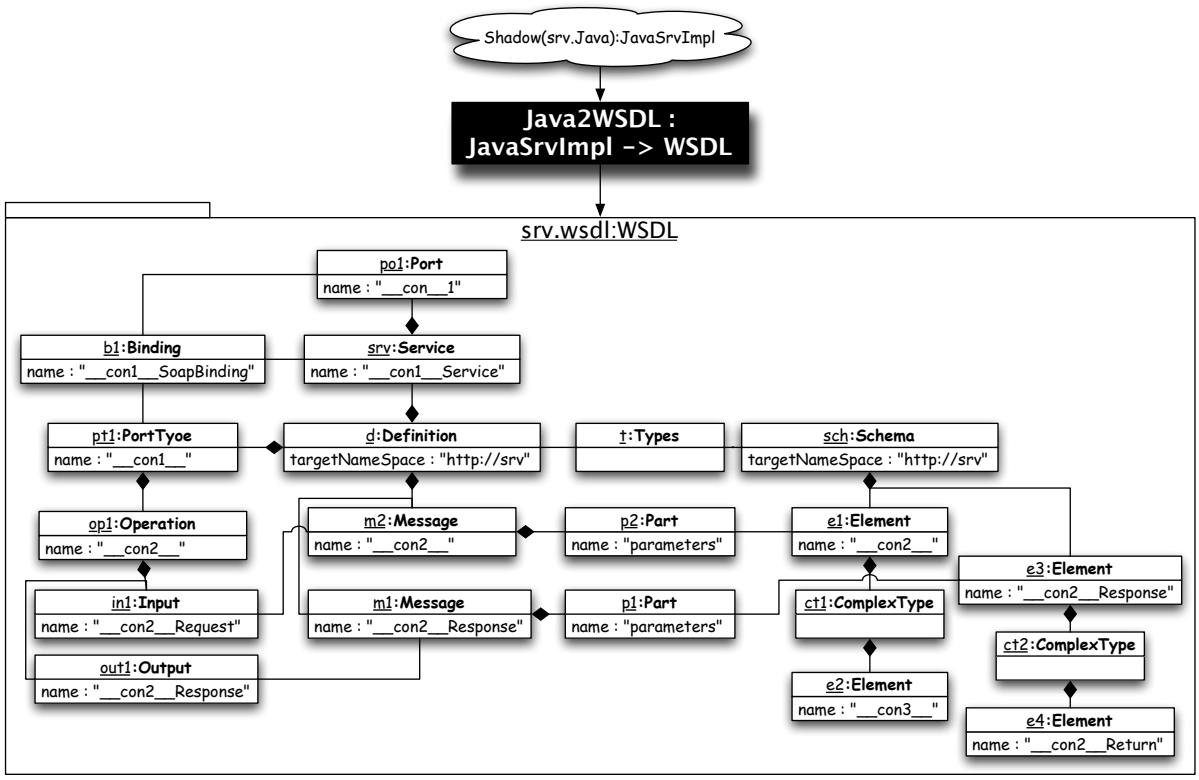


Figure 4.8: Transformation of Java Shadow to obtain WSDL Shadow

4.1.4 De-Shadow

The deShadow operation, listed in Algorithm 4, performs the opposite operation of the shadow algorithm. That is, it scans through the shadow artifacts, extracting the patterns for concept IDs from the attributes of model elements (an attribute can contain more than one concept by means of concatenation). For each concept ID found, it fetches its value from the concept pool, and replaces it with the value.

In this algorithm, attributes are allowed to have multiple concepts. The function matchConceptID takes the value of an attribute in the shadow model, and matches the concept ID pattern against it. This results in a list of concept identifiers found in the attribute value. Each concept ID is then replaced by its value obtained from the concept pool.

Algorithm 4 deShadow S

Input Shadow Model $S = (\mathfrak{C}, \mathfrak{A}, \mathfrak{R}, \mathfrak{T})$ and Concept-Pool $\mathcal{CP}$
Output deShadowed Model M

```
1: function deShadow(S, CP)
2:     M ← Clone(S)
3:     for all s contained in M do
4:         for all a_i ∈ A(s) do
5:             id[1..n] ← matchConceptID(a_i)      ▷ extract concept ID patterns in a_i
6:             v ← a_i
7:             for j ← 1 to n do                   ▷ for all concepts IDs found in a_i
8:                 cv ← CP.getConceptVal(id[j])
9:                 if cv ≠ v then
10:                    v ← replace(v, id[j], cv)   ▷ replace the conceptID pattern with its
   concept value, fetched from the concept pool
11:                end if
12:            end for
13:            if v ≠ a_i then
14:                ♦(s, i, v)                      ▷ Update the attribute's value
15:            end if
16:        end for
17:    end for
18: return M
19: end function
```

4.1.5 Synchronization Process

The idea of propagating model dependencies using shadow models relies on the presumption that, under the course of the transformation, the unique concept IDs in the shadow artifact are not subject to manipulations that make them unrecognizable in the resulting target shadow model. In other words, the essence of transformations for which this methodology is applicable is re-organization of concepts. Consequently, the attribute values of model elements should only be subject to a category of re-writing operations that do not dismantle concept IDs. For example, the transformations are allowed to concatenate two concept values, or add a prefix (or a suffix) to a concept. Operations such as shuffling the characters of a concept, cutting some of the letters or anything else that does not preserve the concept IDs are not directly permitted. This, nonetheless, is not a major limitation for two reasons. On the one hand, concepts are, ideally, the most prim-

itive and finest grained pieces of information in a model. With that perspective, a wide range of meaningful transformations are expected to simply re-organize these quanta of information into different encapsulating data types, rather than tearing them apart. A transformation with such behavior is called a non-mutilating transformation (formally defined in the following). On the other hand, it is possible to work around these limitations. The general methodology to enable such anomaly cases of string re-write operations is to provide post-synchronization fix-up transformations to produce the desired effect of these rules.

Definition 29 *Monotonic transformation* T *is said to be non-mutilating if* $\blacklozenge(\square \mapsto \nu) \xrightarrow{\mathsf{T}} \blacklozenge^*(\square \mapsto p_i \nu p_i')$.

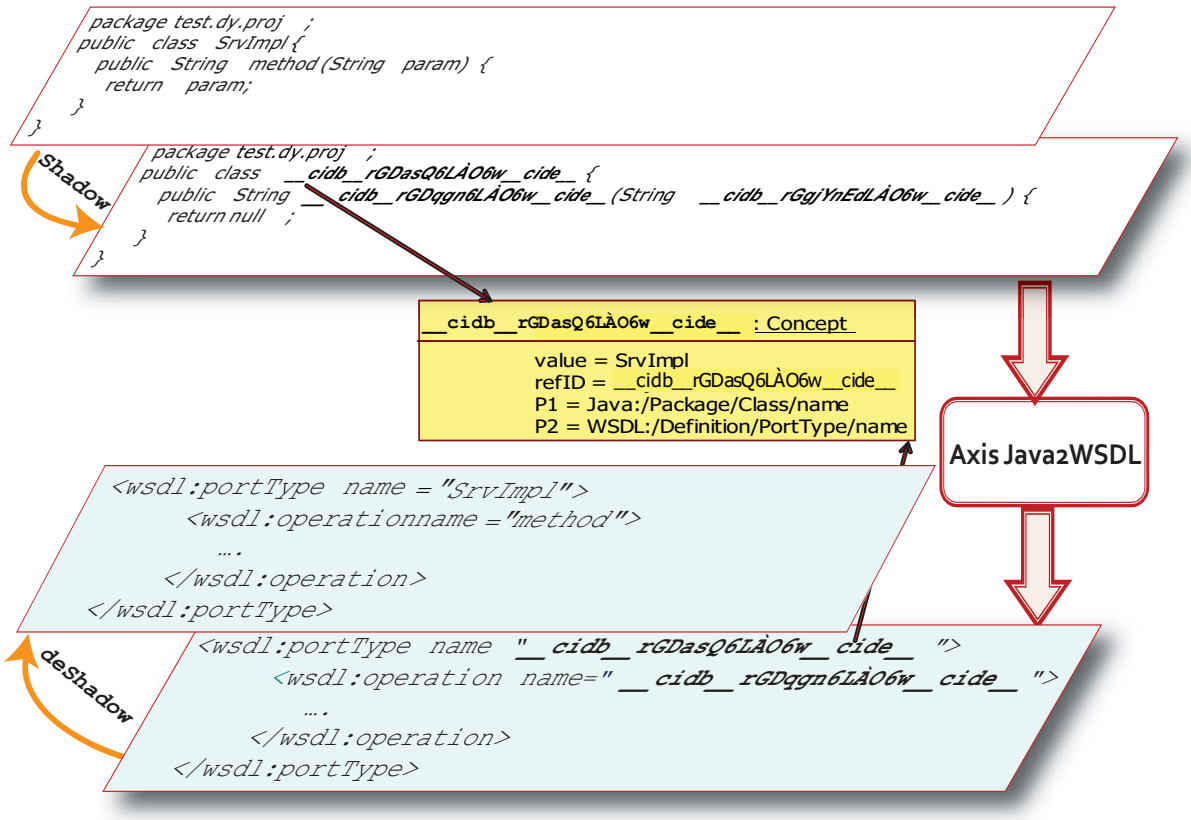


Figure 4.9: Synchronization by Shadow

As a result of Conceptualization and exploiting Shadow models, we can ensure that the source of a model transformation and its product, the target artifact, are entangled recip-

rocally using their mutual concepts. These concepts are stored in the central concept pool along with other concepts in the system, and each is individually and uniquely identifiable by a universal ID. When an attribute of an element in either the source or the target model the object of a change, essentially concepts that represent said attribute are updated in the concept pool. The affected concepts are also tagged as modified, and the time-stamp of the latest change is also recorded in the corresponding entry of those concepts in the concept pool. Two strategies are conceivable for propagating the values of updated concepts to the other artifacts that carry those concepts. The first strategy is to disseminate the changes to all relevant artifacts as soon as they occur. This needs maintaining a list of referencing artifacts for each concept entry in the concept pool. The second strategy is that the synchronization be carried out lazily, that is, each artifact is updated only when it is opened or it is focused on by the user. The outline of the synchronization process that is composed of three Phases, is listed below.

Change Propagation Process

```
Phase A: Conceptualization
  Create Models from Artifacts by abstracting to the canonical representation
  Extract Model Concepts by conceptualization
  Store concepts in Concept Pool
  Create Shadow Models

Phase B: Artifact Generation
  apply the transformation on  source shadows
  deShadow the generated shadow artifact

Phase C: Change propagation
 Upon Change in Source or Target:
  Update related concepts in the concept pool
   For all modified concepts
    Get the locations of all referencing elements
   If synchronization strategy is immediate
      push the changes to all impacted elements enumerated in the concept pool
   else
      render the changes when the affected artifacts are opened or focused on
```

In the first phase of the process above, we set the stage for incremental model synchronization by abstracting the involved software artifacts to a canonical modeling notation. Then we conceptualize the resulting models, and assign unique IDs to each identified concept. The first phase concludes by creating shadow artifacts. The shadow models are denoted in the canonical modeling representation. To be able to use them with the transformation, we need to convert them back to the original artifact's format. For example, the

shadow model created for the abstracted Java code needs to be converted to the textual code representation of a Java class so as to be readable by the Java2WSDL transformation. In the second phase of the process, we execute the artifact generations on the created shadow artifacts, thereby obtaining the shadow models for target artifacts. Subsequently, we run the deShadow algorithm to convert the target shadow to the desired target artifact. The third phase of the process is triggered whenever a change operation in one model raises the need for re-synchronization. The framework traces the modified elements to their concept entries in the concept pool, and updates the corresponding entries in the concept pool.

The synchronization operation needs to render the affected artifacts with the updated concept values. The re-invocation of deShadow for each artifact routinely achieves the desired result, for it fetches the values of all concepts in the shadow artifact from the concept pool, and replaces the concept identifiers with their values, which results in updating the modified ones. It is legitimate to dispute that this last step of the synchronization, i.e., deShadow, is not incrementally performed since it is essentially replacing all concepts, thus, its intrinsic impact is the model in its entirety. Such argument, although valid in theory, does not impair the liveness and responsiveness of the synchronization process in practice as the deShadow operation, in effect, is about the same order of complexity as simply saving and loading artifacts. Nevertheless, the framework also offers full incrementality by maintaining a list of changed shadows in the Concept pool and using the entries of referencing model elements in the concept pool as depicted in Figure 4.9, and only performing deShadow on those entries. The tradeoff, however, is making the concept pool larger and more complex.

Figure 4.9 illustrates the synchronization of the Java2WSDL example utilizing the Shadow/Transform/deShadow process. On the top of Figure 4.9, lies the Java source code, which is the input artifact of the Java2WSDL transformation. The *Shadow* operation encapsulates the following steps in the order given: First, the abstraction of the Java code into the canonical format (Figure 4.5). Second, the conceptualization of the resulting model. Third, performing Algorithm 3 on the abstract model to create its shadow model. Finally, serializing back the resulting shadow model into the Java code format. The result is the shadow code, which, as portrayed in the figure, is structurally identical with the original code in the segments that are relevant to the transformation, modulo the values of the concepts which are replaced by their unique identifiers. Specifically for this transformation,

the bodies of the methods are ignored by the transformation, hence no manifestation thereof in the abstract models, and consequently, neither in the shadow code.

The shadow code is used as the input to Java2WSDL, yielding the shadow WSDL. To obtain the target model, Algorithm 4 is run over the target shadow (illustrated in Figure 4.9 as the *deShadow* arrow). Any two related elements in either sides of the transformation, e.g., the name of the Java *class* and that of the WSDL *portType*, refer to the same entity in the concept pool and thus have the same value, because the concept IDs obtained by looking at the same locations of these elements in their shadows are identical.

It should be noted that the update synchronization aided by concepts and shadow models is two-way, even if the original transformation is unidirectional. This is one of the major benefits of the proposed approach. The synchronization engine only exploits the artifacts, their shadows and the implicit correspondences denoted in the concept pool. No further invocation of the original artifact generator is required in this process. There is no distinction between the source and target of the transformation after it is applied initially. It is therefore possible that the target of the original transformation becomes the source of model synchronization, i.e. the updates that need to be propagated are made to the model that was the product of the transformation.

A special case that deserves further attention is when an updated attribute carries more than one concept. It can be represented by concatenation of conceptualized values and prefixes (or similarly suffixes). For example an attribute can comprise a fixed prefix, the first name, a hyphen–which is another invariant segment–and the last name of a person. The first name and the last name are the conceptualized and variable segments of the attribute value, whereas the title and the hyphen are constant. When the value of such attribute is updated, the system identifies the concepts that are modified and extracts the new segmental values corresponding to those concepts. To detect the altered concepts, the attribute value is screened against its counterpart in the shadow model. The fixed segments appear in both sequences and are used for aligning concept IDs and their segmental values.

4.2 Insertion and Deletion

4.2.1 Propagation of Insertion Induced Changes

Unlike Update, Insertion and Deletion are structure altering changes. The propagation of insertion and deletion changes for an arbitrary model transformation can be multifaceted and complex. This complexity can fortunately be tamed by assuming that the transformation is homomorphic and monotonic (as defined in Chapter 3. The former is to guarantee that the transformation is not sensitive to any particular instance model in its domain, but rather it transforms them all uniformly. The latter requires the change operations to have similar effects on both sides. These two properties seem to be valid for a wide range of model transformations used in practice..

Model elements, according to Definition 1, have containers, which can be inter-dependent across multiple models. In other words, the dependency of element types can be viewed as dependency between the containers of those types. This interpretation of dependency links implicitly requires that insertion of an element to one container (only) result in addition of elements in its inter-dependent containers. We assume that Insertion (and similarly Deletion) homogeneously results in Insertion (and respectively Deletion) type of changes in other inter-related containers. We refer to this property of transformations as *monotonicity*. If insertion of an element results in update or deletion type of changes in the target model, then the transformation is non-monotonic.

Furthermore, we assume that introducing an element into a container follows a uniform pattern of insertions that is independent to the current state of the model (e.g., to the number of elements inside said container). For example, the addition of the third parameter to a method ought not trigger a different pattern of changes in its inter-dependent containers on the WSDL side, than does adding second parameter. We call this property of transformations *continuity* and such transformations are called *continuous*.

Although these two assumptions may seem too restraining, in practice they are in compliance with many artifact generators. In fact, the space of transformations that common relational frameworks such as QVT, TGG, Tefkat etc. are capable of expressing are also uniform and, for the most part, monotonic. For non-monotonic and/or non-uniform transformations, the explicit definitions of the transformation rules that happen to violate either of these two assumptions have to be known.

μ-Templates Overview

The proposed methodology for the propagation of Insertion, for uniform and monotonic transformations, involves deliberately injecting each container in the source shadow model with a dummy placeholder element called a μ-template. When these mutated shadow models are thereafter used as input to the artifact generation process, the μ-templates are transformed and instantiated as target artifacts. More specifically, using μ-templates, target artifacts are generated with an additional hypothetical new element into each of the source's containers. When an actual insertion change to a container in the source model takes place, the added element replaces the available μ-template in the container; this μ-template is *consumed* into an actual element in the source artifact. To accommodate future insertions, a new, unsubstantiated μ-template is subsequently created.

Conceptually, in this process we *a priori* assume the possibility of having an additional element to be inserted in the future for each container, and reserve in advance the appropriate structure and space (i.e., the μ-template) for contingent elements in the container. Upon need, we use these reserved places for adding a new element to containers by updating the values of their attributes and rendering them as visible.

Consuming the reserved space of μ-templates, per se, prohibits adding more elements to the container in the future; simply because each container only has one extra space. Therefore, to continue supporting insertion of new elements into the container, it is necessary to create a new reserved space before consuming the available μ-template. The uniformity assumption enables us to provide a new μ-template by simply duplicating the old one and assigning new concepts to it.

Injection of μ-Template

To better demonstrate how μ-templates are utilized to propagate Insertion, utilizing μ-templates, we proceed with the example scenario of synchronizing an insertion of a method argument to the Java side with its corresponding WSDL model. The steps of the process are illustrated in Figures 4.10, 4.11, 4.12 and 4.13. The first step is injecting μ-templates into the containers of the source artifact. To disguise this amalgamation from the user and make the synchronization as transparent as possible, μ-templates are, in fact, injected into the Shadow models right after their creation. Shadow models, as discussed, closely mirror

the structure of an artifact and are kep invisible from the user.

Algorithm 5 μ-Template Injection

Input Shadow Model $S = (\mathfrak{C}, A, \mathfrak{R}, T)$ and Concept-Pool $\mathcal{CP}$
Output Shadow Model S

1: **function** $\mu - \mathtt{Inject}(S, \mathcal{CP})$
2: **for all** $m \in \uplus S$ **do**
3: **for all** $C_i \in \mathfrak{C}_m$ **do**
4: $m' \leftarrow \blacktriangle(m, C_i, T_m(C_i))$ ▷ Insert a new element in container C_i of element e of model M
5: **for all** $a_i \in A(m')$ **do**
6: $\mathtt{cid} \leftarrow \mathcal{CP}.\mathtt{addNewConcept}(\mu, \mathtt{Addr}_S(m'), i)$
7: $\blacklozenge(m', i, \mathtt{cid})$
8: **end for**
9: $\mu - \mathtt{Inject}(m', \mathcal{CP})$
10: **end for**
11: **end for**
12: **return** S
13: **end function**

The details of injecting μ-templates into shadow models is listed in Algorithm 5. An extra element is injected in each container in the shadow model, and for each attribute of these elements a new, special concept is added to the concept pool. Figure 4.10 shows (the shadow of) an abstracted Java code for a web service on the right hand side. On the left, the same Java model is shown after it is populated by μ-templates. As the figure illustrates using dashed lines, in every container in the model, an extra element is injected.

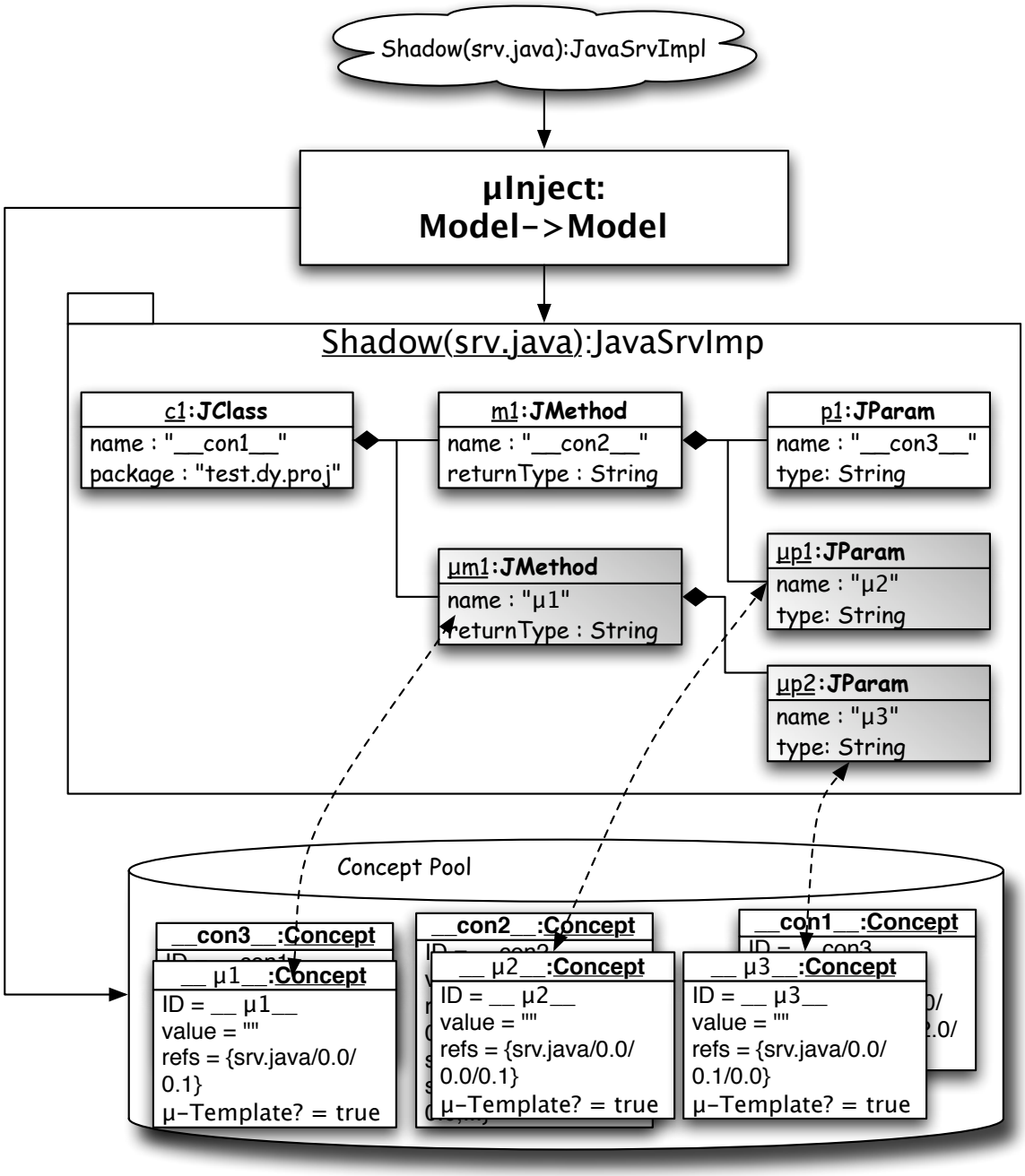


Figure 4.10: Injecting µ-templates

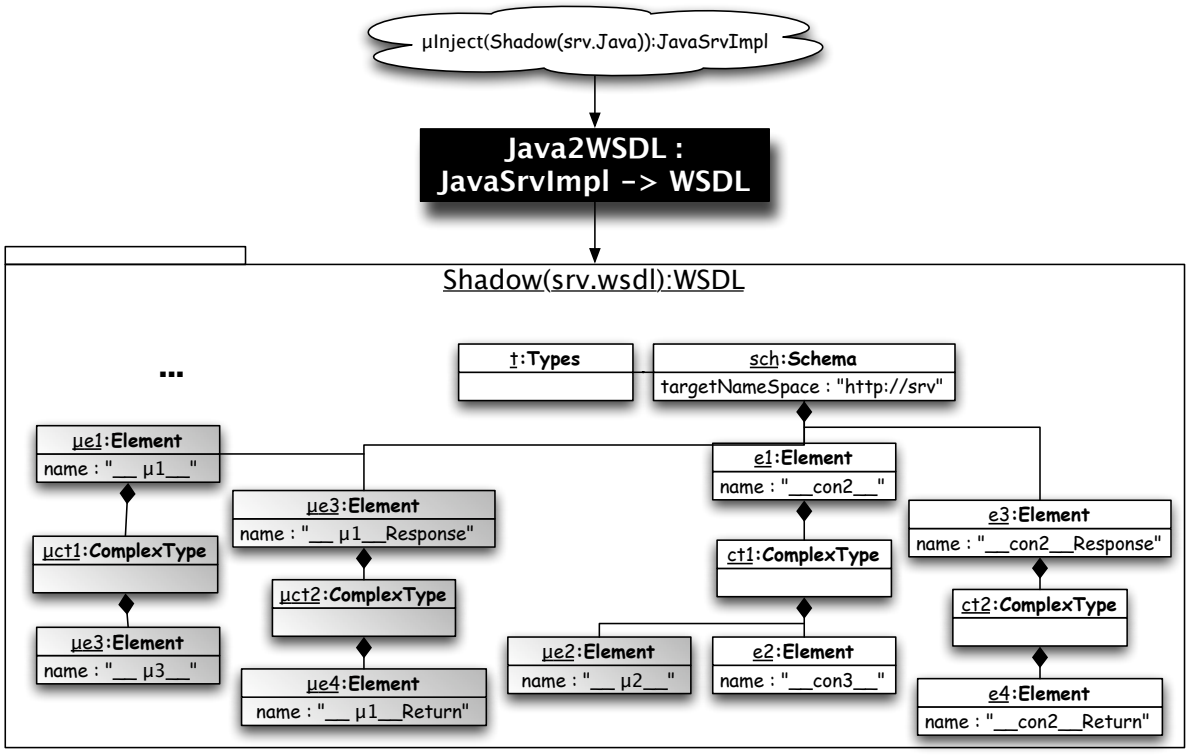


Figure 4.11: Transformation of µ-templates

Transformation of µ-Templates

As stated previously, the shadow of the source artifact is used, in lieu of the artifact, to generate the target of the transformation. The same steps are also involved for synchronizing Insertion changes, Figure 4.11 shows the shadow model of the Java abstraction populated with µ-templates, and its resulting WSDL model which includes µ-templates in several places. µ-templates are discerned from normal elements by having all of their enclosed concepts marked in the concept pool. A model element that hosts a µ-template concept is a µ-element. µ-elements do not manifest in the target model. This requires an extra step to filter µ-elements before rendering the output by deShadow.

Consumption of µ-Templates

Figure 4.12 illustrates the steps involved during the insertion of a new element, a process

we refer to as *consuming* a μ-template. When an element is inserted in one of the containers of the source artifact, the first step is to conceptualize the new element, i.e., capturing the concepts that appear in the new model element. This is realized through consuming the μ-element that is provided for the insertion of a new element in the container. Algorithm 6 presents the details of consuming μ-templates. The μ-element is turned into a normal element by essentially unmarking its μ-template concepts in the concept pool. When consumed, μ-template concepts' IDs remain intact. The synchronization framework, when encounters the IDs of such concepts in other models, deduces that they belong to a recently consumed μ-template concept. The container of the consumed μ-templates need to be populated by new μ-templates to allow for further insertion of model elements. Therefore a new μ-template is created and injected into the container.

Algorithm 6 μ-Template Consumption

Input Shadow Model $\mathcal{S} = (\mathfrak{C}, \mathcal{A}, \mathfrak{R}, \mathcal{T})$, Model $\mathcal{M}$, Element e, Container c, Element Type $\mathcal{T}$, Concept pool $\mathcal{CP}$
Output Shadow Model $\mathcal{S}$

1: **function** μ-CONSUME($\mathcal{S}$, $\mathcal{M}$, e, c, $\mathcal{T}$, $\mathcal{CP}$)
2: $\mu \leftarrow last(\mathcal{S}/Addr_{\mathcal{M}}(c))$
3: $\mu' \leftarrow \mu - Inject(\mathcal{S}/Addr_{\mathcal{M}}(e), c, \mathcal{T}(c))$ ▷ add a new μ element for further additions
4: **for all** $a_i \in \mu$ **do**
5: $\mathcal{CP}.updateConcept(a_i, \mathcal{A}(last(e.c)).i)$ ▷ Update concept's values of μ-template concepts in the pool
6: $\mathcal{CP}.setNextμPointer(a_i, \mathcal{A}(\mu').i)$ ▷ Point to the new μ-template concepts in the concept pool
7: **end for**
8: **return** $\mathcal{S}$
9: **end function**

Propagation of Changes

As discussed in Subsection 4.1.1, in order to synchronize the target model with the modified source model, the deShadower is reapplied on the target shadow model. The μ-template concepts are converted to normal elements in the concept pool at consumption time, i.e., when the source model was modified by an insertion. Therefore, the deShadower, when reinvoked over the target shadow model, would no longer discard these elements and

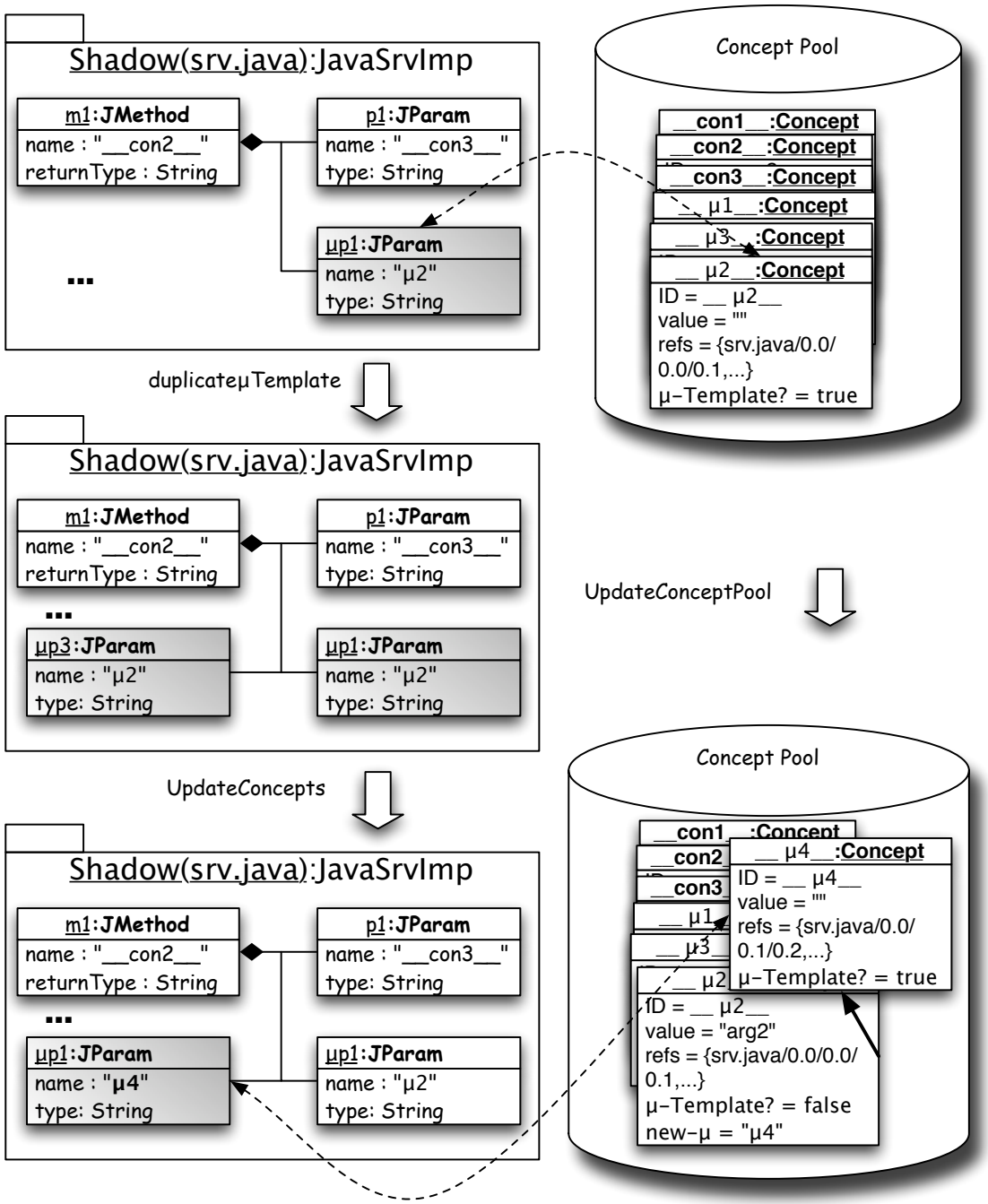


Figure 4.12: µ-template Consumption

will render them in the output model, as demonstrated in Figure 4.13. Similarly to the source model, a new μ-template needs to be placed in the container to enable further insertion of elements in it. Unlike the source model, providing a new μ-element to the target model involves a few more steps than simply duplicating the former μ-element and adding brand new concepts to it. In particular, the dependency links between the new μ-template and the one that was just inserted in the source model have to be established by making them reference the same concepts. To that end, we need to find out the concept IDs that are assigned to the newly created μ-template in the source model when the old μ-template was consumed. As usual, our medium for communicating such information is the concept pool. Therefore, this can be enabled by providing pointers in the entries of consumed μ-template concepts in the concept pool to the new concepts created for the new μ-template. For example, in Figure 4.12, when the μ-template is consumed (and its value is updated to *"arg2"* in the concept pool) it points to the concept associated with the newly created μ-template in its container. The target side is only aware of the consumed μ-templates' IDs, since the new ones were not present in the model at the time of transformation. However, the deShadow algorithm follows these pointers for each concept to reach the new μ-template's concept IDs.

Algorithm 7 enumerates the steps outlined above for propagating insertion changes using μ-templates.

Algorithm 7 $\mu - \mathsf{Filter}$

1: **function** $\mu - \mathsf{Filter}(S, \mathcal{CP})$
2: $\quad R \leftarrow \mathsf{clone}(S)$
3: $\quad$ **for all** $s \in \uplus S \wedge \mu - \mathsf{elem}(s) \wedge \neg\mu - \mathsf{elem}(s/..)$ **do**
4: $\quad\quad$ **if** $\bigwedge_{\mu_i \in A(s)} \mathcal{CP}.\mu_i.new - \mu \neq \bot$ **then**
5: $\quad\quad\quad s' \leftarrow \mathsf{clone}(s)$
6: $\quad\quad\quad$ **for all** $a_i \in A(s)$ $s@a_i = \mu_i$ **do**
7: $\quad\quad\quad\quad s' \leftarrow \blacklozenge(S/s'@a_i \mapsto \mathcal{CP}.\mu_i.new - \mu)$
8: $\quad\quad\quad$ **end for**
9: $\quad\quad$ **else**
10: $\quad\quad\quad S \leftarrow \blacktriangledown(R/s)$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: **end function**

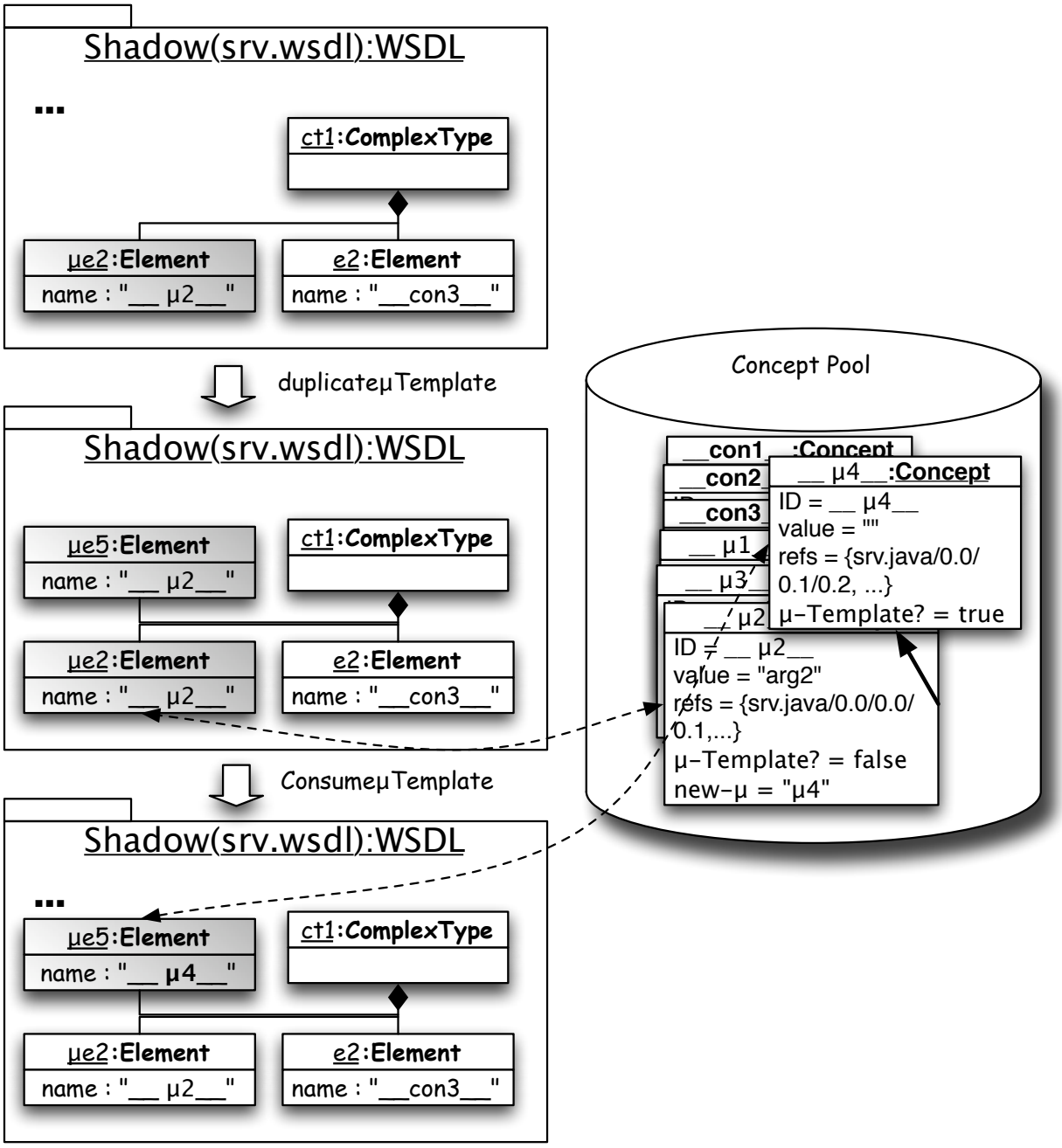


Figure 4.13: Propagating Insertion

Using μ-templates, we have reduced the problem of propagating Insertion to an already solved problem of propagating Update.

4.2.2 Propagation of Deletion

When an element is deleted, all of its enclosed concepts are tagged as *deleted* in the concept pool. Deletion is handled simply by hiding model elements whose concepts are tagged with the special flag *deleted* in the concept pool. For example, when the parameter of a method is deleted from the source code, all the concepts of its corresponding element in its abstract model are tagged as *deleted* in the concept pool. When synchronization is carried out on other models, the framework simply conceals the elements, any of whose concepts are flagged as *deleted* in the concept pool.

When considering containment, propagating deletion changes raises some ontological issues. More specifically, we need to recognize the existential causality relationships between the model elements in order to properly identify the elements that have to be purged as a result of a deletion change. The semantics of containment relationship provides useful guidelines for such reasoning. Briefly, deletion of a containment results in purging all its contained elements and, consequently, their constituting concepts. Therefore, when performing inter-model change propagation, all elements whose any concepts are flagged as *deleted* will likewise be flagged as *deleted*.

4.2.3 Dependency Inference

It is possible to conceive elements in the target model of a transformation that do not contain any conceptualized information from the source model, yet their existence is caused by the existence of some other elements in the source model. Such dependency relationship is called purely existential, inasmuch as it can only be violated by structural changes, namely, Insertion and Deletion. An example of elements with this kind of relation is the ComplexType elements in the type definition segment of the WSDL models. For each Method in the Java model, the Java2WSDL transformation creates a pattern comprising a ComplexType node, which has no attributes. As such, it remains unaffected by any Update changes applied to the corresponding method.

However, the propagation of the Insertion changes using μ-templates needs some adaptation to cope with these purely existential dependencies. As explained, μ-templates are recognized–and on that basis filtered in the output target model–by the virtue of their enclosing concepts being tagged as such in the concept pool. But, an element such as ComplextType does not share any concepts with its originating μ-template in the source model, even though its *raison d'être*[1] is the transformation of that element from the source model. This existential relationship posits a twofold problem for Insertion. On the one hand, these elements cannot be recognized as μ-template on the target models and will incorrectly pass through μ-Filter into the final target model. On the other hand, when their corresponding μ-template is consumed in the source model, they are not properly duplicated along with other μ-template bearing elements for future additions.

The following presents a portion of type definition in WSDL that corresponds to a μ-template of a method in the source.

$$\begin{aligned}
&\mathsf{type} = (\langle\langle \mathsf{elem1}, \mathsf{elem2} \rangle\rangle, \varnothing, \varnothing, \mathsf{Type}) \\
&\mathsf{elem2} = (\langle\langle \mathsf{ct2} \rangle\rangle, \langle \text{"}___^{-}{}_2___\text{"} \rangle, \varnothing, \mathsf{Element}) \\
&\mathsf{ct2} = (\langle\langle \mathsf{elem3} \rangle\rangle, \varnothing, \varnothing, \mathsf{ComplexType}) \\
&\mathsf{elem3} = (\varnothing, \langle \text{"}___^{-}{}_2___\text{"} \rangle, \varnothing, \mathsf{Element})
\end{aligned}$$

When rendering the output models, μ-Filter detects elem2 and elem3 as μ-templates, but is not able to do so for ct2 because it does not have any μ-template concepts. In order for the resolve these issues, we make some additional rules as to when an element should be filtered out as a μ-element. An element is considered a μ-element, if:

1. It has no concepts and all of its children are μ-elements.
$$\frac{\forall n \in \uplus m. \quad \mu - elem(n) \wedge A(m) = \varnothing}{\mu - elem(m)} \ (1)$$

2. It is contained by a μ-template element.
$$\frac{\exists p. \quad m \in \uplus p \wedge \mu - elem(p))}{\mu - elem(m)} \ (2)$$

[1]Reason for being

3. One of its attributes consists of a μ-template concept.

$$\frac{\exists a \in A(m).\quad \exists c \in \mathcal{CP}\quad \mu - \mathsf{Template?}(c) \wedge a \hookrightarrow c}{\mu - \mathsf{elem}(m)} \ (3)$$

These rules help deduce whether an element is a purely existential μ-template by the aid of their enclosing or enclosed μ-templates. However, if the subject element lacks such parents or children (e.g., if ct2 were contained by type and had no children) then the framework is not able to detect it. Such situations have to be explicitly specified for each transformation, and they have to be taken care of as post-synchronization fixes. In the first case above, the purely existential μ-element is consumed if any of its children are consumed. In the second case, it is consumed when its parent is consumed and in the third case it is consumed when all its μ-templates are consumed.

A transformation is defined to be μ-*preserving* if all elements dependent on the μ-templates injected to the source side can be detected using the three rules outline above.

4.3 Complete Picture for The Running Example

Figure 4.14 conjures up all the components of the synchronization framework we have so far described into a workflow. On the right side of each step, the corresponding interim artifacts for the Java2WSDL example are illustrated. This setup workflow is deployed in lieu of the original transformation. The Abstractor and DeAbstractor are domain-specific units that translate artifacts from their original format (e.g., Java code or XML) to the adopted unifying model format (e.g., EMF).

As a result, the target and source artifacts are interlinked through shadow models, which also enable the addition of new elements to both sides. In the life cycle of these artifacts, this workflow is executed only once. Once the artifacts are in place and in adjunction with the concept pool, the algorithm listed in Algorithm 8 carries out the synchronization of artifacts for a given composite change.

In the synchronization algorithm, the applied change to the source model is first factorized, resulting in a sequence of atomic change operations. For operation based systems, changes are already represented as such as composite. For state based systems, change factorization can be performed using Algorithm 1 or 2 or a variant thereof. This step, in fact,

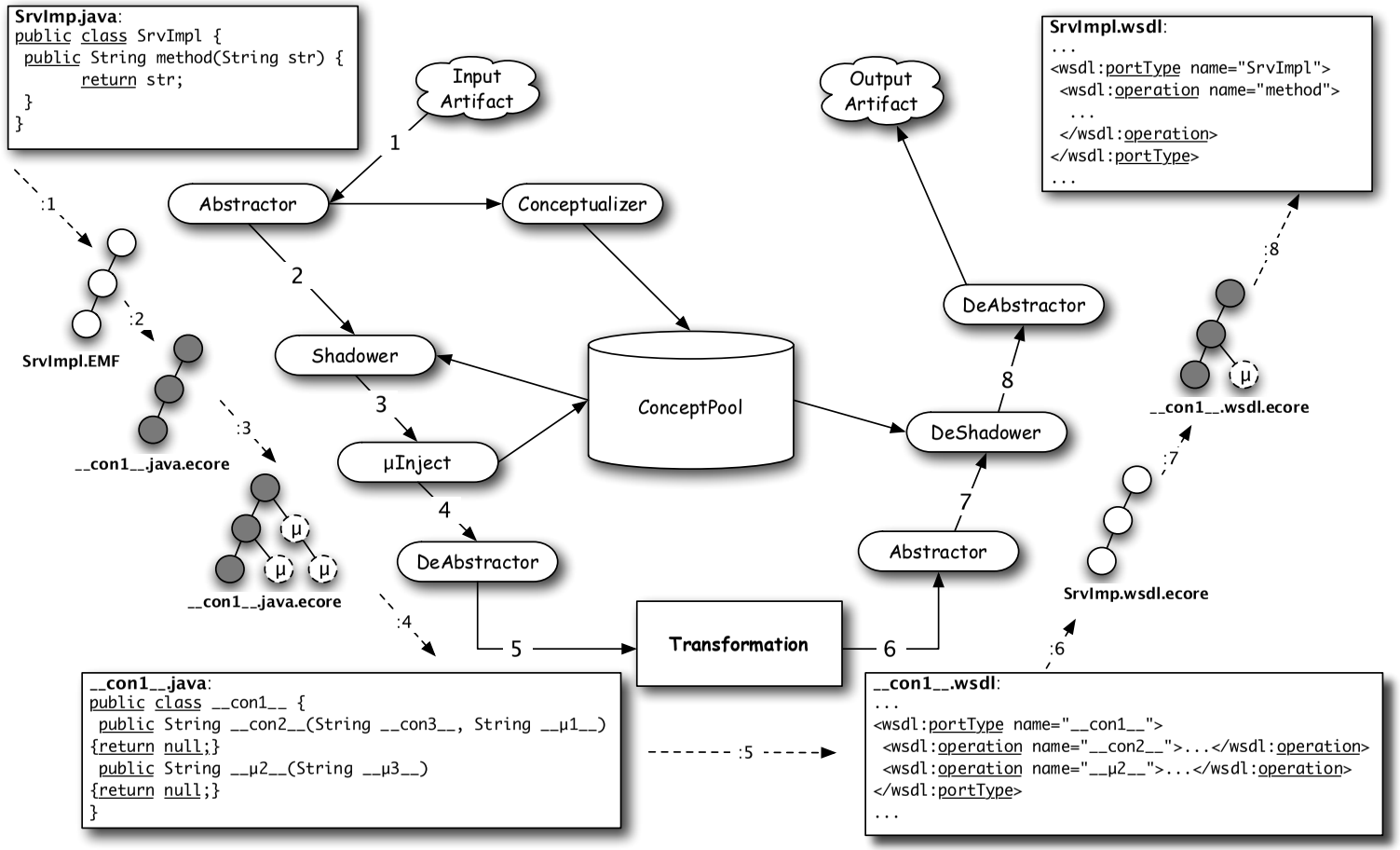


Figure 4.14: Black-Box Synchronization Process

Algorithm 8 Full Synchronization Process

1: **function** $\mathtt{Sync}(M, \Delta, N)$
2: let Sh_M = Shadow of M and Sh_N = Shadow of N
3: $[\delta_1, ..., \delta_n)] \leftarrow \mathtt{factorize}(\Delta)$
4: **for** $\delta_i(p), i \leftarrow 1..n$ **do**
5: $s \leftarrow Sh_M/Addr_M(p)$
6: **if** $\delta_i = \blacklozenge(p, i, v)$ **then**
7: $\mathtt{cid} \leftarrow s@i$ ▷ obtain the concept id by looking at the same attribute in the shadow element
8: $\mathcal{CP}.\mathtt{updateConcept}(\mathtt{cid}, v)$
9: **else if** $\delta_i = \blacktriangle(p, i)$ **then**
10: $T_i \leftarrow \pi_2(\pi_1(T(p)))$ ▷ type of the ith container. Types are the second element of signature returns
11: $\mu - \mathtt{Consume}(Sh_M, M, p, i, T_i, \mathcal{CP})$
12: **else if** $\delta_i = \blacktriangledown(p, i)$ **then**
13: **for all** $e \in (\uplus(\mathtt{last}(s/i))) \cup \{\mathtt{last}(s/i)\}$ **do**
14: **for all** $\mathtt{cid} \in A(e)$ **do**
15: $\mathcal{CP}.\mathtt{setDeleted}(\mathtt{cid})$
16: **end for**
17: **end for**
18: **end if**
19: **end for**
20: **return** $\mathtt{deShadow}(\mu - \mathtt{Filter}(\mathtt{deleteFilter}(Sh_N)))$
21: **end function**

performs differentiation for state-based environments between the two versions of altered models. More specifically, the changed elements can be detected by pair-wise traversal of models and their shadows, comparing the concept values with attribute values to detect update changes, and scanning the positions of μ-template elements in the model to detect insertions.

Given the sequence of atomic changes, the synchronization is carried out for each change operation individually. For updates, the value of altered concepts in the concept pool are updated. For insertions, the type of the container is extracted from the metamodel, which is passed to μ-Consume (Alogrithm 6), along with the address of the container in which the new element is inserted. For the case of deletion, the elements are not purged from the concept pool or the shadow model; instead, all the concepts' belonging to deleted elements are marked as *deleted*, which proscribes them from being rendered in the output. Finally, the last stage is to synchronize the target model. The elements with a deleted concept ID are first filtered from the shadow model. The result is passed to another filter which purges the unused μ-template elements, and as described in the previous section, also duplicate consumed μ-templates and replace their concept IDs according to the pointers set in the concept pool during $\mu - \mathsf{Consume}$. The filtered shadow model of the target artifact is then deShadowed to obtain the consistent target model. A DeAbstraction step is also performed in case the ultimate schema for the target model is different than our unifying model format.

4.4 Properties of the Black-box Synchronizer

4.4.1 Termination

Theorem 4.4.1 *If* MM *and* NN *allow only cycle-free containers, then the black-box model synchronization* Sync *process always terminates for any uniform and monotonic transformation* $\mathsf{T} : \mathcal{L}(\mathsf{MM}) \longrightarrow \mathcal{L}(\mathsf{NN})$, $\mathsf{M} : \mathsf{MM}$, *and an arbitrary change sequence* Δ.

Proof. Algorithm 8 for a given pair of original and modified model, first factors out the change sequence (line 3) presented in Algorithm 1, , which is a terminating algorithm because it is a simple recursive traversal of the containment hierarchies of the models and results in a change sequence consisting of a finite number of atomic operations. The body

of the loop between lines 4 and 14 also consist of constant-time operations (as well as the call to $\mu - \texttt{Consume}$ listed in Algorithm 6, which is evidently terminating). The loop in the segment that handles the deletion case (lines 14-18) also performs a constant-time operation on a finite number of elements (descendants of the one being deleted), hence always terminates, too. The last-line, calls the *deShadow* procedure listed in Algorithm 4, while applying two simple filters to the model. The canonical listing for *deShadow* is also a simple one-pass traversal of the target shadow model, which is guaranteed to terminate. □

4.4.2 Transformation Chains

Although we have established the termination of the synchronization process for a single model transformation, in general, the termination property might not hold when several model transformations are chained together to posit a multi-lateral consistency relationship between models. Figure 4.15 outlines such a situation when three models are involved. Models M1, M2 and M3 are in an initially consistent state. This is characterized by relationships $\mathsf{R}_{1,2}$, $\mathsf{R}_{2,3}$ and $\mathsf{R}_{3,1}$. These relationships can be realized by model transformations. The homeostasis of the outer circle can, however, be disrupted by a change occuring to any of the models. In the figure, model M1 is altered by change Δ_{11}, which modifies it to M_1'. Because the modified model is no-longer consistent with M_2, the synchronization process is triggered, which translates the source change Δ_{11} with respect to relationship $\mathsf{R}_{1,2}$ to a change applicable to M_2, viz., Δ_{21}, which converts M_2 to M_2'. Similarly, to reconcile R_{23}, the synchronization process translates Δ_{21} to Δ_{31}, which modifies M_3 to M_3'. This, however, may violate the $\mathsf{R}_{3,1}$ relationship, hence another change is generated by Sync, i.e., Δ_{12}, which migrates M_1' to a second modified state M_1''. This chain of events spirally continues until a multi-lateral consistency state is re-established (e.g., in two rounds for the scenario depicted in Figure 4.15).

Figure 4.16 exemplifies a concrete scenario of synchronization cycles in transforamtion chains. A very simple model element is defined to represent variables. The first model is a variable in the *under score* notation. This is transformed to what is known as the *Camel Case* convention, by `Underscore2CamelCase` (`U2C`), which essentially removes the underscore character and capitalizes the first letter of its succeeding part, and concatenates it with the first part. The camel case notation is then converted to so called *Hungarian* notation

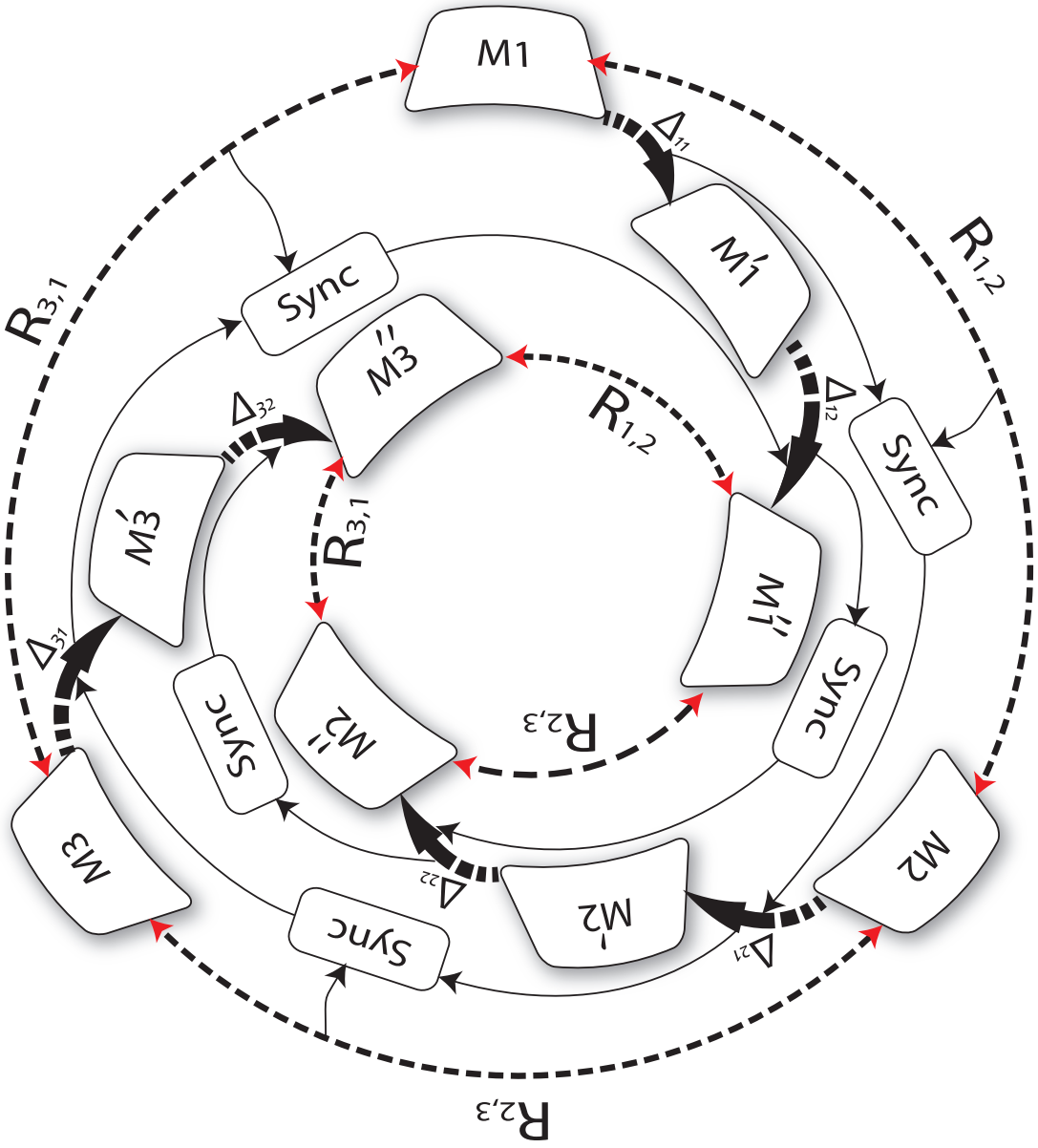


Figure 4.15: Spiral Synchronization of Dependency Cycles

by transformation `CamelCase2Hungrain` (`H2C`). The second transformation concatenates the first letter of the type name of the variable to the camel-case variable name after capitalizing its first letter. A third transformation, viz., `Hungarian2Underscore` (`H2U`) reverts the hungarian notation back to the underscore notation. This is done by removing the

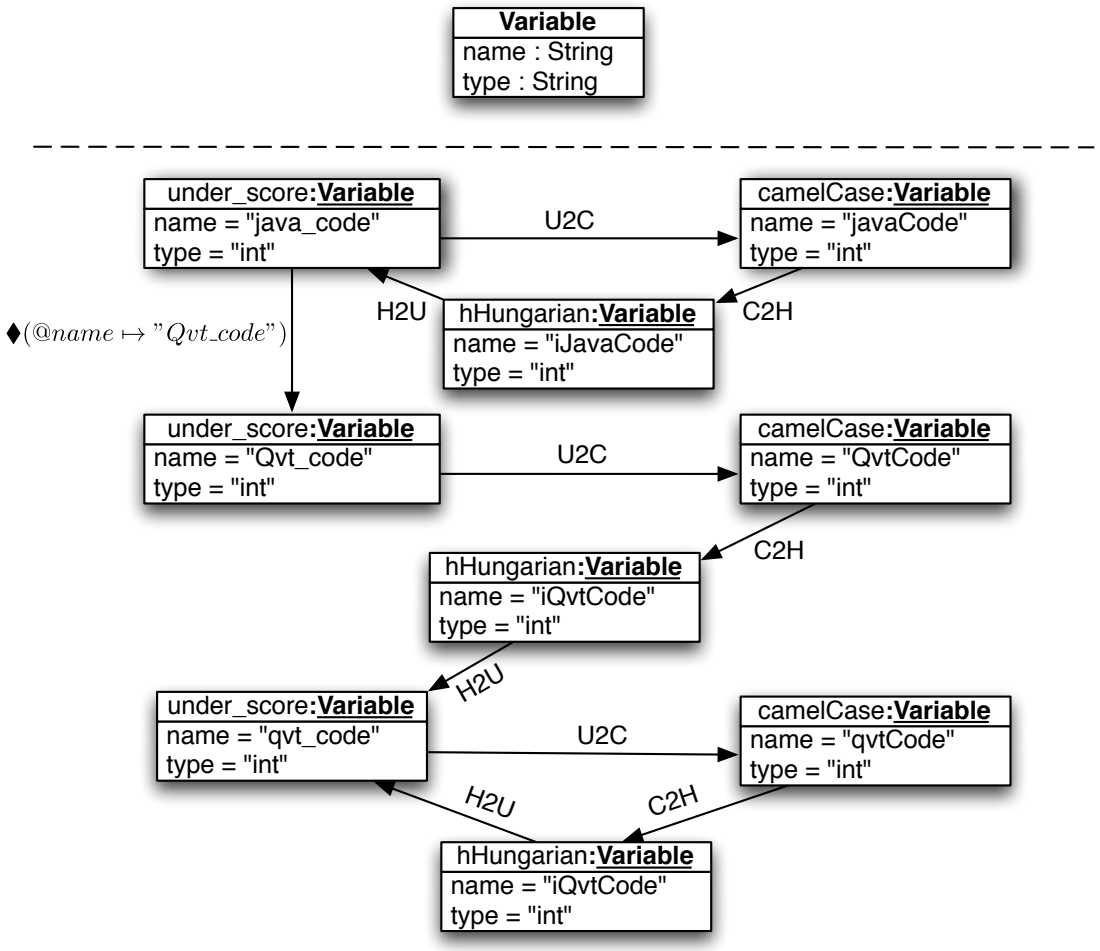


Figure 4.16: Mutli-step consistency establishment

first letter–which represents the type in the Hungarian notation–, making the second letter lowercase, splitting the string at the second occurances of an upper case character, interspersing an underscore there, and, finally, concatinating with the second half whose first character is converted to lower-case. The QVT-OM implementation of these three mappings are presented in the following.

As the example of Figure 4.16 illustrates, the first round of applying the transformations does not reach a consistent state, however applying them for another round reconciles the inconsistencies. The important question to answer for these cases is whether the synchronization chain in such a situation ever terminates. There is always a possibility of

```
intermediate class Variable {
  name : String;
  type : String;
}
mapping Variable::Underscore2CamelCase : Variable {
  init {
    var usPos := self.name.indexOf('_');
    var firstPart := self.name.substring(1,usPos);
    var secondPart := self.name.substring(usPos + 1, self.name.length()).firstToUpper();
  }
  type := self.type;
  name := firstPart + secondPart;
}
mapping Variable::CamelsCase2Hungarian : Variable {
  type := self.type;
  name := type.substring(1, 1) + self.name.firstToUpper();
}
mapping Variable::Hungarian2Underscore : Variable {
  init {
    var i := 3;
    while (self.name.substring(i,i) >= 'a'
        and self.name.substring(i,i) <= 'z') {
      i := i + 1;
    };
    var firstPart := self.name.substring(2, i - 1);
    var secondPart := self.name.substring(i, self.name.length());
  }
  type := self.type;
  name := firstPart + '_' + secondPart;
}
```

oscilating between two states, thus being trapped in a non-halting loop. A pragmatic meta-rule imposed by a control strategy can aid in these situations. One obvious example is to put a hard limit on the number of allowed synchronization cycles, and allow inconsistencies to exist after attempting certain number of attempts.

It should be noted that the above example is not the kind of transformation that the black-box model synchronization methodology readily handles, as it contains attribute mutilating mappings. These are typically handled by suplementing the synchronizer with *ad hoc* post-fix rules, and often utilizing specialized conceptulization schemes. For this particular example, the variable name in the first model can be conceptualized into two different concatenated concepts. The shadow value for `name` in the first model is thus `"cid1_cid2"`. The two concepts' values represented by `cid1` and `cid2` are `"java"` and `"code"`, respectively. This is converted to `"cid1Cid2"` which is no longer recognizable by the de-shadow operation. Assume that we have two fixup rules that respectively coverts this value to `"cid1cid2"` and performs the required capitalization after de-shadowing. Similarly, we will have `"icid1cid2"` as the name of the third variable. Updating the first model updates `cid1` from `"java"` to `"Qvt"` in the concept pool. Performing the Sync operation on each of the two models followed by the fixups, results in the same values as shown in the first cycle of Figure 4.16. Achieving consistency in this case requires two additonal fixup rules for the first and second models, that is converting to lower-case the value of the first concept after the de-shadow phase.

To avoid termination issues, a system using Sync for black-box synchronization of multiple models adopts the general strategy of using post-fixes for reconciling remaining descrepencies, thereby avoiding performing multiple updates to the concept-pool for a given change. This ensures that the argument made for the termination of a single transformation synchronization also holds valid for the multiple-transformation and cyclic cases.

4.4.3 Soundness

One constraint on Sync is that the transformation should be non-mutilating. It means that on the target side of a non-mutilating transformation, attribute values would be recognizable by lexicographic matching. That is to say what the transformation does to its input model is essentially restructuring and re-organizing the information encapsulated in the attribute values. The black-box synchronization methodology is able to capture

the logic of this restructuring without explicit knowledge of the transformation rules. The following theorem posits the soundness of this synchronization scheme for non-mutilating, uniform and monotonic transformations.

Theorem 4.4.2 (Soundness) *For a given uniform, monotonic, non-mutilating and* μ*-preserving transformation* $\mathsf{T} : \mathcal{L}(\mathsf{MM}) \longrightarrow \mathcal{L}(\mathsf{NN})$ *and a composite change* Δ*, the black-box synchronization is sound, that is,*

$$\forall \mathsf{M} \in \mathcal{L}(\mathsf{MM}) \;\; \mathtt{Sync}_{\mathsf{T}}(\mathsf{M}, \Delta) = \mathsf{T}(\Delta(\mathsf{M}))$$

Proof Correctness of the $\mathtt{Sync}$ process for update changes is evident based on the discussion in the Subsections 4.1.2, 4.1.3 and 4.1.4. We briefly re-iterate the gist of the argument here. Consider model elements $\mathfrak{m}$ and a change $\blacklozenge(\mathfrak{m}@\mathfrak{a}_i \mapsto \nu)$. Because the transformation is assumed to be monotonic, update changes translate to a number of updates on the target side of the transformation. Without loss of generality we can assume that the change maps to a single update change on the target. Let $\mathfrak{n}$ be an element on the target side affected by the translation of the update, that is, $\blacklozenge(\mathfrak{n}@\mathfrak{a}_j \mapsto \mathfrak{p}\ \nu\ \mathfrak{p}')$. Let S and S' be the shadow model in the source and the target side, respectively. The Shadow Algorithm 3 ensures that $\mathsf{S}/\mathtt{Addr}_{\mathsf{M}}(\mathfrak{m})@\mathfrak{a}_i = \mathtt{cid}$. Because T is non-mutilating, $\mathsf{S}'/\mathtt{Addr}_{\mathsf{N}}(\mathfrak{n})@\mathfrak{a}_j = \mathfrak{p}\ \mathtt{cid}\ \mathfrak{p}'$. As presented in Algorithm 8, $\mathtt{Sync}$ updates the corresponding concept entry–found by looking at the shadow model of the source model–in the concept pool, that is, after the change, $\mathcal{CP}.\mathtt{getConceptVal}(\mathtt{cid}) = \nu$. Thus de-shadow (Algorithm 4) updates the value of $\mathfrak{n}$ to $\mathfrak{p}\ \nu\ \mathfrak{p}$.

Due to monotonicity, $\blacktriangle(\mathsf{M}/\alpha) \xrightarrow{\mathsf{T}} \blacktriangle^i(\mathsf{N}/\beta_i)$. Due to Lemma 3.3.14, all the μ-templates in M/α are localized in elements contained by N/β_i. Algorithm 7 will thus render each element N/β_i as visible because they all contain μ-templates that are consumed. Similar argument establishes that deleted elements are correctly filtered by the $\mathtt{Sync}$ process.

□

4.5 Complexity

To assess the complexity of the synchronization we define the notion of change complexity, that is, the number of change operations performed on the model. This implicitly means

taxing update operations over other kinds such as simple model traversals, when estimating the runtime complexity. The complexity projected in this way however remains in acceptable correspondance with reality for most pragmatic cases. Furthermore, many of the algorithms here that have full-model traversals (e.g., deShadow in Algorithmalg:deshadow) are presented as such for the sake of clarity. As the concept-pull maintains for each concept a list of referencing model elements, it is possible to realize these algorithm in an efficient way that seeks right to the affected element, thus eliminating the need for full model traversal. The framework's implementation in Eclipse which has been the subject of our experiments presented in the next section has used several kinds of optiimzations such as the ones described.

Theorem 4.5.1 *The black-box synchronizer has worst-case complexity of* $O(|\Delta|)$ *and space complexity of* $O(|\mathsf{M}| + |\mathsf{N}| + |\Delta|)$.

Proof. Space overhead is basically associated with the shadow models on both sides of the transformation and the space occupied in the concept pool which is linear with respect to the size of the models. The number of changes performed by Algorithm 8 is also proportionate with the size of the factorized change sequence processed in line 4, as for each kind of change a constant number of change operations are performed.

□

4.6 Experiments and Evaluation

4.6.1 Experiment Setup

For the evaluation of the proposed framework, we have designed a prototype which we applied for the incremental synchronization of models in the Eclipse Web Tools Platform (WTP). WTP encompasses several types of software artifacts each having different format and schema. Furthermore, it extensively uses transformations for converting these artifacts to one another. One such transformation, which we also have used throughout this chapter for presentation, and also for conducting our assessments, is Java2WSDL. Java code is a textual file that is parsed into an Abstract Syntax Tree (AST), and is compiled into a binary class file. In contrast, WSDL is an XML document that conforms to the WSDL schema,

which is specified in the XML schema format. Moreover, because of its extensibility type definitions, which are XML schema elements, the type definition part of WSDL conforms to *XML Schema for XML Schema* (the meta-metamodel of XML).

Our experiments were conducted on a personal computer featuring Intel®Core™2 processor and 2.00 GB of main memory, with ample disk space made available for virtual memory utilization. Eclipse 3.4 (Ganymede) equipped with WTP 3.0–with no unnecessary plugin installed–was run on Microsoft®Windows™XP Service Pack 3.0. No other major application or service was running simultaneously during the course of experiments. Lightweight programmatic instrumentation and time measurement (console I/O essentially) was used. The time spent on UI interactions needed for initiating several processes (e.g., launching the transformation wizard) was excluded.

4.6.2 Results

Figure 4.17 compares the execution time of synchronization against that of re-transformation of increasingly larger Java classes, i.e. with more methods, and their resulting WSDL files. The Y (i.e., elapsed-time in seconds) axis is outlined in logarithmic scale. This graph also shows the framework's setup time, that is, the time spent to initialize the framework and create the shadow models. It is evident that the time cost of synchronization is almost independent of the models' sizes; contrary to the time required for regeneration, which acutely increases as the size of the input model grows. From the graphs in Figure 4.17, we observe that a Web Service system exposing close to ten thousand methods (a quite excessive figure for practical systems) is taking approximately 1000 seconds to regenerate all the models using all the available transformation rules (top line), while the time to perform the initial setup and to incrementally synchronize the models is approximately 6 seconds and 8 seconds respectively.

The Java2WSDL transformation, in fact, ranges over multiple input files. This situation arises when one Java class references another class through methods' argument types or return types. In such cases, Java2WSDL also creates type definition for the referenced class in the WSDL file. To further assess the performance of our synchronization framework, we utilized this aspect of the transformation as an instance where the complexity of the subject transformation also progressively increases. Figure 4.18 shows the result of synchronization for hierarchies of multiple Java beans and their corresponding WSDL. The setup time

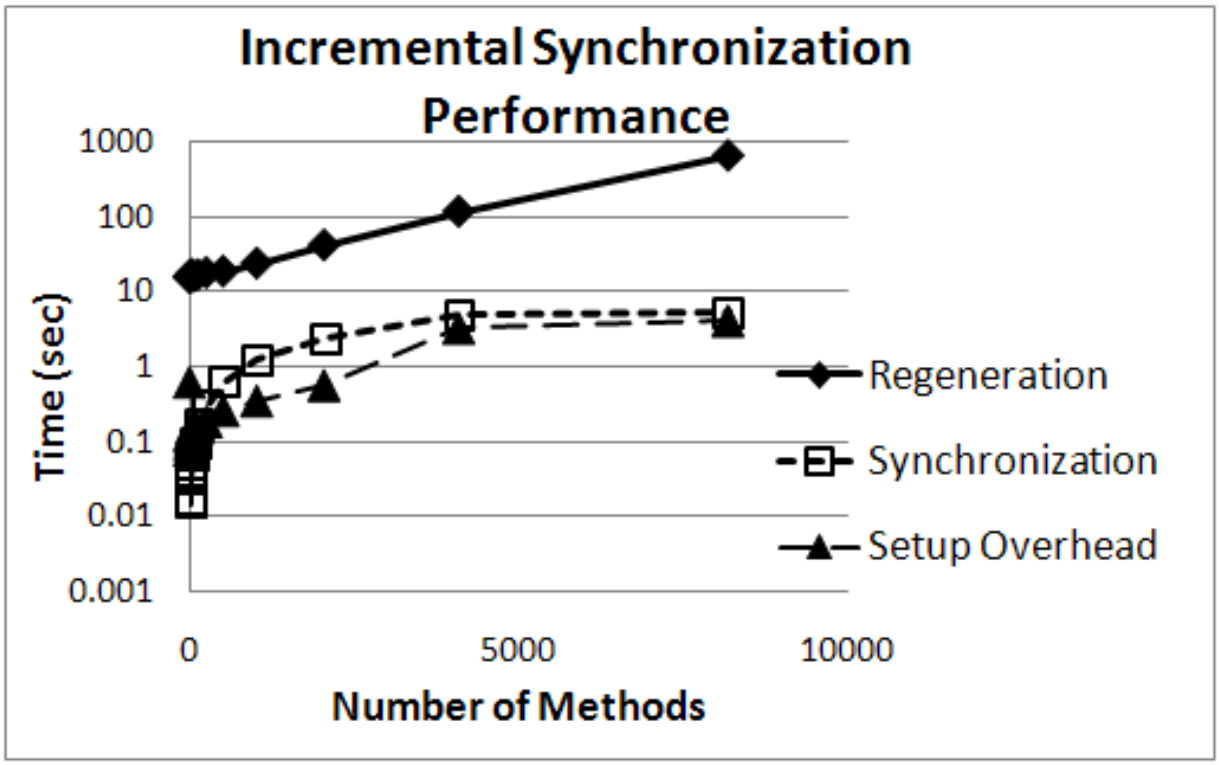


Figure 4.17: Performance Evaluation(Log Y-Axis)

is higher than the previous experiments, albeit still markedly faster than regeneration. This difference is predominantly due to the relatively high overhead of file I/O in Eclipse workspace, and the fact that the experiment with multiple beans naturally involves many more such operations. The results in this figure indicate that for a system composed of 512 beans the complete regeneration takes approximately 300 seconds while the setup time for the incremental synchronization process takes approximately 150 seconds. Once the setup process is complete, then synchronization due to insertion and undate induced changes takes almost constant time of less than 10 seconds. Nevertheless, the setup process takes place only once at the beginning, so, in effect, it does not slow down the synchronization phase.

The extra space required by the shadow models is reported in Figure 4.19. As the figure depicts, the space overhead of shadow models and μ-templates tends to be on the same order of magnitude of the size of the models, hence it does not pose any limitations

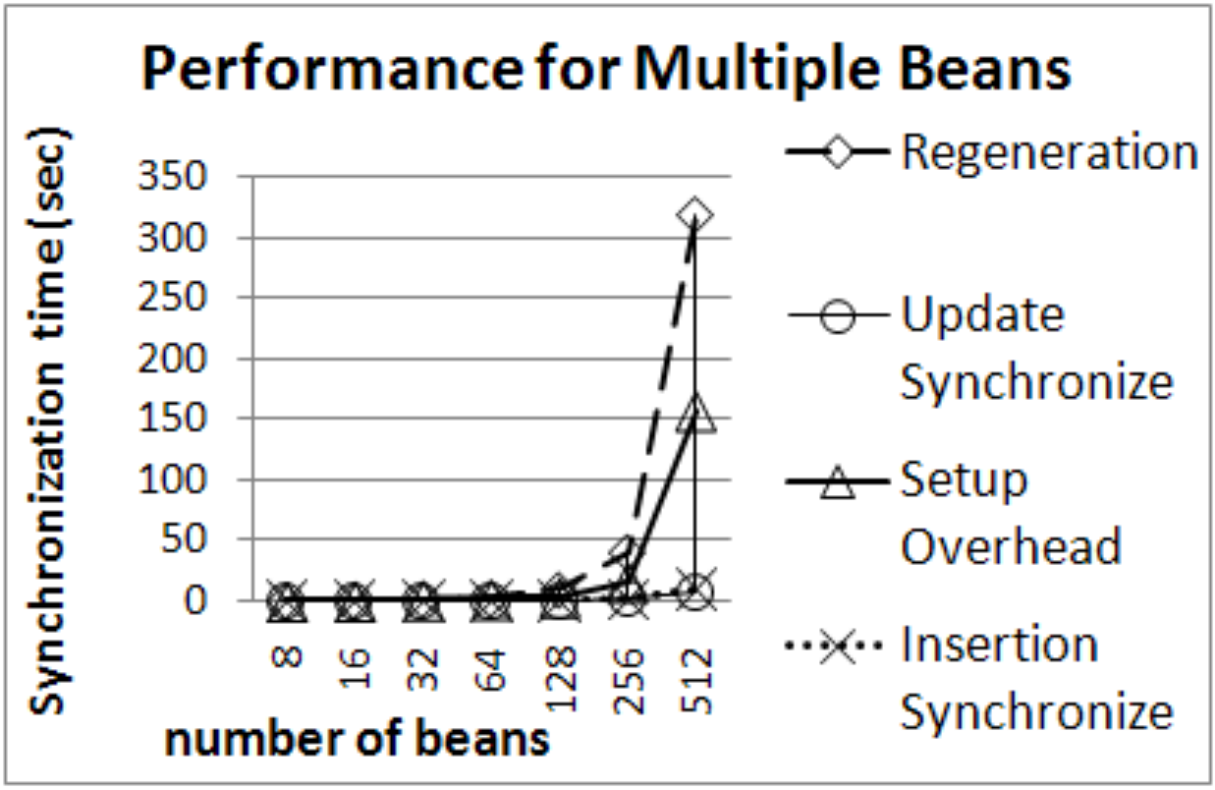


Figure 4.18: Performance for multiple beans

on the system.

4.7 Application and Integration

In this section we discuss the integration of the presented synchronization framework with the Eclipse Web Tools Platform (WTP) [80]. WTP is a collection of bedrock technologies aimed to facilitate the development and maintenance of Web applications in general, and Web Services specifically, within the Eclipse ecosystem [24]. WTP features a comprehensive Web Services subsystem that allows developers to develop, assemble, deploy, invoke and test Web Services in a manner congenial with the needs of large scale enterprise projects. Two primary trajectories for developing web services has been envisaged: bottom-up and top-Down development. In the former approach, users start from the implementation of

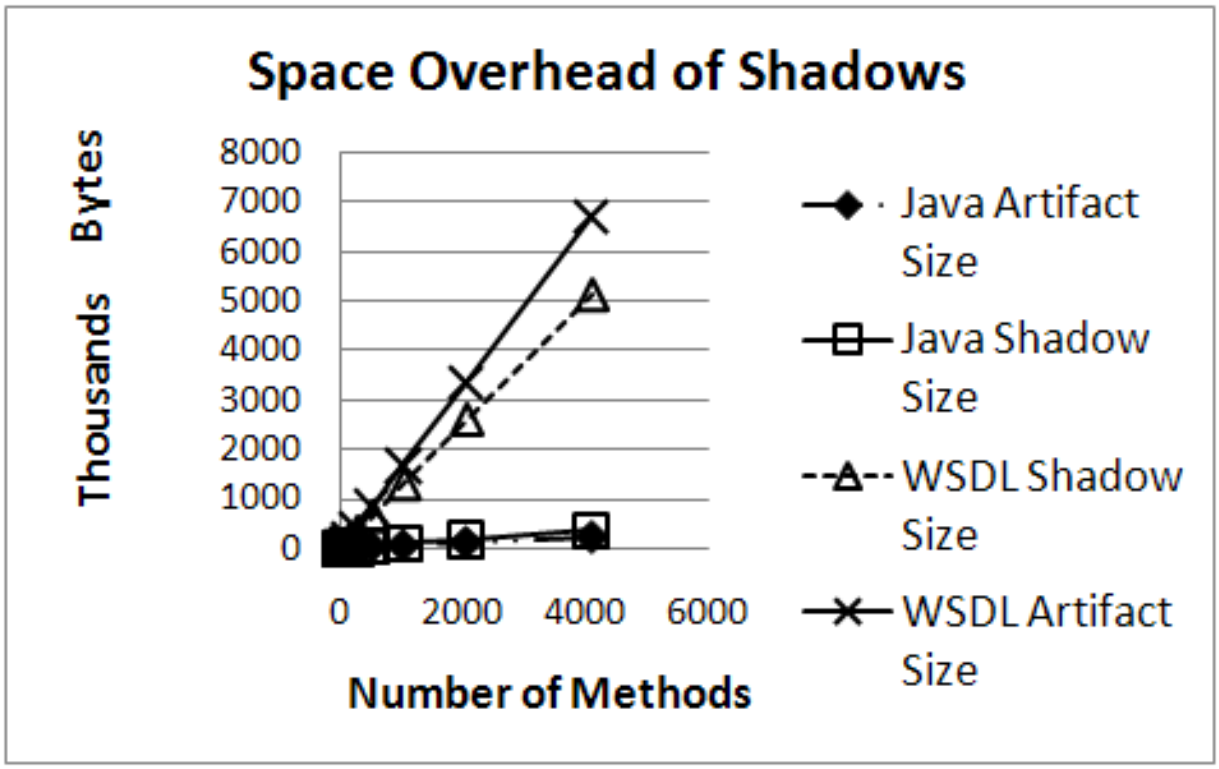


Figure 4.19: Shadow files space overhead

a web service, usually in Java, and apply the provided chain of transformation to create the service level artifacts, that is to say, WSDL, Web Services Deployment Descriptor (WSDD), metadata for application server configuration, and other artifacts. In contrast, in the top-down path, another chain of transformations is applied on the description of web services (i.e., WSDL) to generate stubs for Java implementation. Moreover, in either of the approaches, further software artifacts–e.g., Java code for unit testing and service invocation clients as well as Java Server Pages (JSP) proxies–can be added to the already eclectic mix of interdependent models.

WTP utilizes a third party tool from Apache Axis project [7], Java2WSDL, in order to generate WSDL files from Java code. The developer initiates the generation process by choosing the *Bottom-Up Web Service Creation* strategy in a WTP UI wizard and then guides it through for further customization. In addition to the time needed by the transformation, going through this elaborate, multi-step user interface for each modification to

the source code is a cumbersome task for developers. Another advantage attained from deploying live synchronization is the elimination of such redundant UI interactions.

To effectively leverage the proposed synchronization framework in an existing IDE, a number of design decisions, on the basis of the characteristics of the IDE, has to be made. To fully harness the power of the framework, the host IDE has to cater for certain requirements. In the following, we discuss the particulars of each decision point.

Notification Mechanism

Most IDEs provide an even driven environment to some degree. The offerings range from resource and project level events to fine-grained notifications delivered to individual model elements. Fine grained notification can be instrumental for live, on-the-fly synchronization of simultaneously open models. Eclipse Modeling Framework (EMF) provides notification mechanisms of this sort (by means of the Notifier class) which is the root of the type hierarchy in EMF [26]. As such, all EMF objects can potentially emanate EMF Notification messages, when they are modified.

Unifying Metamodel

To deal with the diversity of artifacts, the proposed solution takes advantage of a unifying metamodel. As described, the synchronization process, in a step referred to as abstraction, creates from software artifacts models that are expressed in this canonical, unifying metamodel. The described synchronization algorithm are then performed on these abstract models, which are ultimately converted back to their original format. A light-weight meta-modeling scheme close to the formalism we offered in Section 3 is preferred.

Change Listener

For software artifacts that are not opened in the IDE, a change listener mechanism is needed to notify the synchronization framework of the changes made to the artifacts. Eclipse IResourceChangeListener and several other classes provide this sort of facility. In the absence of this feature, the synchronization should be initiated manually by the user.

Change Factorization

As noted, this solution has adopted an operational view of change. This implies that for a complex change operation, the system should provide details of the constituting atomic change operations and the location of the updated elements for each atomic change. Both Eclipse (for resources) and EMF (for resources as well as in memory Objects) provide such elaborations. For state-based settings lacking this feature, the atomic changes may be calculated in two ways. The first one is the joint traversal of models with their shadows while comparing the values of each attribute with those of their representing concepts in the concept pool. The second one is comparing models with its previous version if it is retained in the system. The first approach is always plausible while the second one can, by and large, be expected to perform better since it requires fewer inquiries from the concept pool.

UI/Editor Integration

Proper integration with editors' user interfaces enables the framework to operate speculatively, that is, it can detect the changes as they are made in the file, even before the resource is saved. The system can thus give the user interactive guides as to the valid ways of editing a model and previewing the impact of each edit before it is persisted in the file.

Concept Pool Choices

We presented the concept pool as a database of concepts. While this abstraction can be directly realized by deploying one of the available object oriented embedded databases (e.g., Apache Derby) other alternatives may be more preferable in some cases. In our implementation, we simply have used an XML file, which persists the in-memory hash maps that indexes the concepts.

Transparency

It is essential to hide the extra resources created by the framework (e.g., shadow models) from users. In Eclipse, the *metadata* directory is the de facto place for storing such information.

Lazy versus Eager Synchronization Strategy

The choice of synchronization strategy between eager synchronization–that is, immediately pushing updates of a concept to all its referencing artifacts–and deferring the synchronization to when affected artifacts are accessed, which we call lazy synchronization, is to be made based on the characteristics of projects. For a project with numerous artifacts the lazy synchronization strategy is preferred, inasmuch as users' span of attention will, in practice, only survey over a handful of resources at a given time. However, cases where models are required to be consistent at all times are conceivable (e.g., in debug mode or in deployed services with hot-replacement capabilities).

Server Deployment and Runtime Issues

In IDEs that support the deployment of generated executables remotely or in local, embedded execution platforms (such as embedded Servlet containers like Apache Tomcat) extra steps have to be taken to gracefully re-deploy the updated models, and/or restart the server with modified configuration files.

Refactoring Framework

If the IDE offers a refactoring subsystem, it is paramount to couple it with the synchronization framework so as to enable the sophisticated, and often semantics, refactoring operations for the updates propagated by the framework. Eclipse for instance, offers the Language ToolKit (LTK) subsystem to handle the refactoring operations in a generic fashion. We coupled our prototype to this framework for enabling refactorings to happen on non-conceptualized, and as such un-entangled, pieces of code in the projects, which still have inter-dependencies with the conceptualized resources that the framework manages.

Chapter 5

White-Box Incrementalization of Transformations

In generative Model Driven Engineering environments, it is common to apply model transformations iteratively and frequently. As we have discussed, the repetitive application of these transformations, especially when they are complex and/or when the source models are copious, can take a significant amount of time. In the previous chapters, we introduced methodologies for the incremental synchronization of transformations by exploiting their generic properties in a *black-box* fashion. However, the black-box approach, albeit relatively inexpensive to employ, is inherently limited. There are transformation patterns whose synchronization requires greater amount of information about their internal structure than what can be elicited using the methods presented in Chapter 4. Such information can still be collected by opening the so called "box" and analyzing the source code of the transformations' implementation. We refer to this methodology of deriving incremental transformations by performing static analysis on the source code of the transformation as *white-box* synchronization.

In this chapter, we discuss a strategy of *incrementalizing* model transformations based on partial evaluation–that is, pre-evaluating parts of programs using input data that are known *a priori*. To this end, a prototypical partial evaluator for Object Management Group's Query, View and Transformation (QVT) Operational language is developed and used to specialize experimental QVT transformations.

Partial evaluation is an established methodology in the area of programming languages

research that is based on the premise that programs can be executed on a subset of input data known *a priori*, so as to generate a *residual* program, whose expressions are, to the extent allowed by the availability of known data, statically pre-evaluated. Thus, the residual program, when executed over the dynamic inputs, does not need to compute the parts of the code that correspond to the known inputs. Performing fewer computations at runtime, residual programs are expected to run faster than their original counterparts [41, 2, 46, 42].

The key idea behind our approach is that model transformations are also a kind of software programs, which get as input instances of a metamodel (e.g., MOF or similarly Ecore). When a transformation is applied iteratively, the altered elements of the source model can be considered as dynamic data, and the invariant fragments as static. The catch is to reduce the number of computations performed when a complex transformation is invoked by pre-computing and storing in a residual transformation the expressions that are not affected by a model change. As a proof of concept, we have developed a prototype of a partial evaluator for a subset of OMG's imperative model transformation language, i.e., QVT Operational Mappings[10]. Transformations denoted in QVT, and in other model transformation languages alike, make extensive use of collection operations to manipulate the elements of input models and their containers, to form the target model. Therefore, our technique primarily focuses on the specialization of these sort of expressions. Our prototype partial evaluator is also implemented in the QVT-OM language. This design decision has two important methodological consequences. On the one hand, it adheres to the general philosophy of model driven engineering, which strives to treat all major software component as models; in particular, the object of our partial evaluator–i.e., QVT transformation–are themselves treated as MOF models, and are manipulated as such. On the other hand, this enables the concept of self-application, that is, specializing the partial evaluator by itself.

5.1 Introduction to Partial Evaluation

Partial evaluation of software programs refers to a pre-execution process whereby parts of the program are pre-computed with values known before runtime so as to yield a new *residual* (or specialized) program equivalent with the original one–in the sense that the

resulting program, when executed at runtime over full set of inputs, will produce the same output as the original program. The partial evaluation process aims to produce a residual program that ideally computes only the parts of the program that correspond to the unknown inputs. Therefore, it is expected to exhibit better overall performance than the original program.

More specifically, let

$$[\![\mathtt{prog}]\!]_{\mathcal{L}}[\mathtt{in}] = \mathtt{out} \tag{5.1}$$

denote that $\mathtt{prog}$ is a program specified in language $\mathcal{L}$ and produces output $\mathtt{out}$ for the sequence of inputs $\mathtt{in}$. The $[\![]\!]_{\mathcal{L}}$ notation indicates that $\mathtt{prog}$ is evaluated as expressions written in language $\mathcal{L}$. Suppose that the first $\mathtt{m}$ inputs of the program (denoted as $\langle \mathtt{k}_1, .., \mathtt{k}_\mathtt{m} \rangle$) are known *a priory*–that is, before the execution time–and the rest of the inputs are unknown (denoted as $\langle \mathtt{u}_1, .., \mathtt{u}_\mathtt{n} \rangle$). A partial evaluator for $\mathtt{prog}$ is a program such as $\mathcal{PE}$, inputs of which are the source code of program $\mathtt{prog}$ in language $\mathcal{L}$, and its set of known inputs. It transforms the input program to a *specialization* of it, referred to as $\mathtt{prog}_{\mathtt{res}}$, with respect to this set of known inputs.

Figure 5.1 depicts the general process of partial-evaluation. Partial evaluation is a form of program transformation as it produces another program as output. Let

$$[\![\mathcal{PE}]\!]_{\mathcal{L}'}[\mathtt{prog}, \mathtt{k}_1, .., \mathtt{k}_\mathtt{m}] = \mathtt{prog}_{\mathtt{res}} \tag{5.2}$$

indicate that $\mathcal{PE}$ is interpreted as expressions of language $\mathcal{L}'$ and processes $\mathtt{prog}$ (written in $\mathcal{L}$) and inputs $\langle \mathtt{k}_1, .., \mathtt{k}_\mathtt{m} \rangle$. The result of this process is a residual program $\mathtt{prog}_{\mathtt{res}}$ with the property that when run on the rest of the inputs it yields the same output, that is to say:

$$[\![\mathtt{prog}_{\mathtt{res}}]\!]_{\mathcal{L}''}[\mathtt{u}_1, .., \mathtt{u}_\mathtt{n}] = [\![\mathtt{prog}]\!]_{\mathcal{L}}[\mathtt{k}_1, .., \mathtt{k}_\mathtt{n}, \mathtt{u}_1, .., \mathtt{u}_\mathtt{n}] = \mathtt{out} \tag{5.3}$$

.

Note that in the general case, the target language of the residual program (i.e., $\mathcal{L}''$) and the language used for the specification of $\mathcal{PE}$ (i.e., $\mathcal{L}'$) need not be the same as the original program, viz., $\mathcal{L}$. In fact, when $\mathcal{L} \neq \mathcal{L}'$ the process involves a concomitant source translation phase.

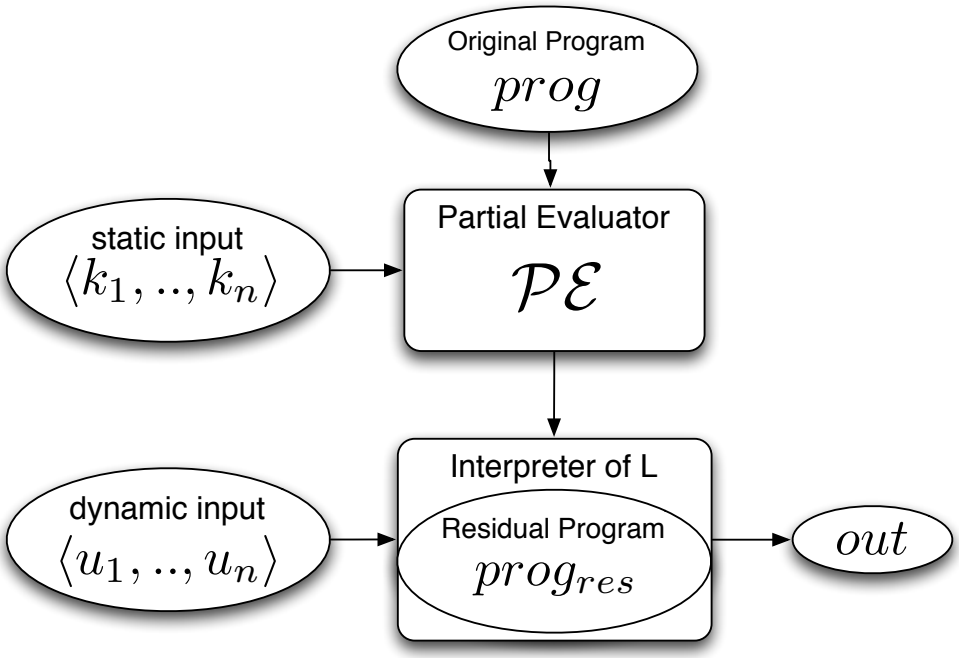


Figure 5.1: Partial Evaluation (adapted from [42])

$\mathcal{PE}$ aims to pre-compute as many of expressions of `prog` as possible at specialization time with respect to static data and merge the pre-computed values with the rest of the program. Because some of the expressions in `prog` are replaced by statically pre-evaluated values in $\texttt{prog}_{\texttt{res}}$, the latter is intuitively expected to perform better than the former.

Partial evaluation usually consists of two phases. During the first phase, which is called *Binding Time Analysis*, the source code of the program is analyzed with respect to the set of known inputs and thereby the constructs inside the source code are annotated as either *static* or *dynamic*. Following this analysis, in the second phase the constructs that are determined to be static are evaluated, starting from the inputs and incrementally replacing each static expression with its evaluated value. Dynamic expressions in contrast are substituted with symbolic expressions that are derived from the values of static expressions and other dependent dynamic expressions. The second phase yields the residual program, which only needs the unknown subset of the inputs of the original program to run. There are two common strategies to carry out these two phases; explicitly and separately in *offline* evaluators, versus *online* evaluation by performing static analysis on the go

along with specialization [42]. Either way, statically computable expressions of the original programs are replaced with pre-evaluated values in the residual program and thus will not be recomputed during runtime. The comparative efficiency of the residual program is thus a direct product of how aggressively can the partial evaluator determine the binding time of intermediate expressions and reduces them to correct specialized expressions.

It should be noted that partial evaluation is generally no gaurantee of performance. In fact, the most naive way of partially evaluating $\mathtt{prog}[\mathtt{k}, \mathtt{u}]$ with respect to $\mathtt{k}$ is to inline the value of $\mathtt{k}$ in a wrapper that simply invokes $\mathtt{prog}$. Partial evaluation may also inflate the size of the program, which in certain execution contexts (e.g., for interpreted languages) may result in relatively poor performance due to the overhead incurred on the parser as a result of code bloat.

There are two particular factors involved in improving performance of residual programs. First, the more precise can the partial evaluator determine the binding time of intermediate variables based on the input division the higher the fraction of expressions which it can statically reduce. A conservative strategy would let many reduction opportunities pass unscathed by failing to recognize that they can be evaluated statically. Second, coalescing statically computed expressions with the rest of the program can be challenging and may require many *ad hoc* strategies whose complexity in part depends on the uniformity and sophistication of the target language. Languages such as Lisp or Scheme have relatively simple and elegant syntactic structures, and as a result lend themselves easily to static analysis processes, including partial evaluation. Some imperative languages (e.g., C++), in contrast, pose daunting obstacles for partial evaluation due to their complex (and sometimes inconsistent) syntax and semantics and multitude of unsafe features that they allow programmers to exploit.

We discuss the white-box synchronization techniques by presenting *QvtMix* a partial evaluator of the Object Management Group's Query, View and Transformation language for Operational Mappings (QVT-OM). QVT-OM provides a hybrid collection of imperative and declarative constructs, and is designed to operate in one direction with no direct support for incremental execution of transformations.

5.2 Introduction to Query View Transformation Operational Mappings

Query, View and Transformation (QVT) is a specification devised and published by Object Management Group (OMG). It defines three related languages for expressing transformations between MOF compliant models. QVT has emerged out of seven initial submissions to a request for proposal by OMG in 2002 to unify the hitherto varigated methods used for expressing model transformations. QVT's key requirements have been:

1. Compositionality of transformations
2. Support for arbitrary domain models specified in a unified meta-modeling notation such as MOF
3. Support for different levels of complexity to be able to express simple and sophisticated transformations
4. Interoperability with transformations denoted in other technologies/languges
5. Provision of a visual, model-oriented notations in the spirit of MDE
6. Support for seamless interchange of models and transformations

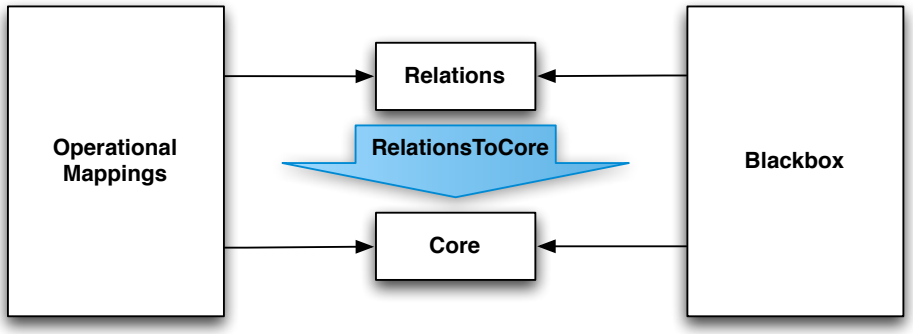


Figure 5.2: Query View Transformation stack of model transformation languages

QVT Relations (QVT-R) provides declarative relations for establishing correspondence between models, as well as validating such correspondences. It defines a graphical notation

(similar to TGG [31]) as well as a textual syntax for the specification of the relations. The relations are type-driven and are qualified using trigger patterns, annotations in the form of `when` clauses that determine the applicability of a relation to model elements. Transformations are defined as relations (in case of QVT-R) or mappings (QVT-OM) between meta-model elements and are then *applied* on instances of metamodels.

The core language is another declarative fragment of the QVT stack, which provides lower-level constructs for model transformations than those of the relation language. The relation language can, in fact, be implemented by mapping its constructs to those of the core language. QVT provides an interface called *black-box* for the invocation of external transofrmations. This provides a venue for extending the language and its standard libraries using general purpose programming languages and take advantage of existing libraries and frameworks.

5.2.1 Model Transformation with QVT Operational Mappings

QVT-OM is an imperative language designed to facilitate the specification of transformations between models denoted in formalisms like MOF or Ecore. The imperative features of QVT-OM offer greater expressive power than the declarative parts of the language suited for creating complex structures. In this section we aim to provide a brief overview of the language and introduce the constructs and features that are essential for understanding the remainder of this thesis.

The abstract syntax of QVT is defined as a UML metamodel using MOF notation. This makes possible treating QVT transformations as models and using QVT to define higher-order transformations. Furthermore, it enables many possibilities for easily developing various kinds of useful meta-transformations and code generators.

The basic construct of the QVT language is a module. Each module contains a transformation, which takes a number of argument. Each argument refers to a model and has a name, a type name, which corresponds to a metamodel, and a direction, which indicates if the model is input, output or both input and output (for in-place, endogenous transformations). The following is an example signature of a stereotypical transformation that maps the input metamodel `MM` to the output metamodel `NN`, which are illustrated in Figures 5.3 and 5.4, respectively.

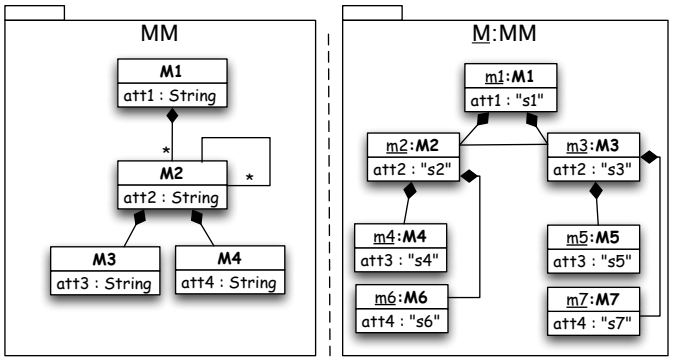


Figure 5.3: Sample source metamodel MM and its instance M

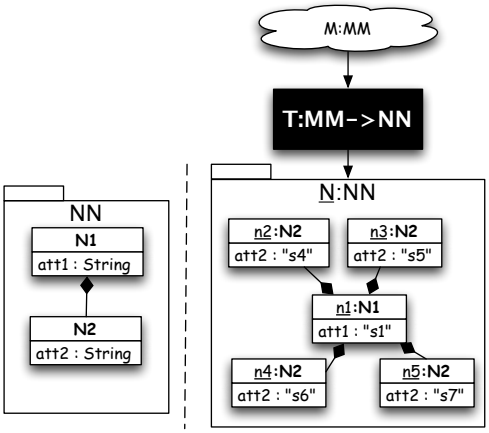


Figure 5.4: Transformation `T` mapping `M:MM` to an instance of the target metamodel `NN`

```
transformation T(in M : MM, out N : NN);
```

Transformations consist of a number of operations, with one being a mandatory entry operation, namely, `main`. QVT-OM defines three different kinds of operations:

1. *Query*: side-effect free, pure functions answering to OCL types of queries on models
2. *Helper*: auxiliary, imperative functions for general purpose computation
3. *Mapping*: uni-directional, type-driven correspondences between input and output models of the transformation

The entry operation is typically used to access the elements of input models, sift them based on their types and other criteria, and invoke some mapping operations on them to obtain target elements. For example, the body of transformation `T` is listed below. It accesses the top level objects of the input model using a built-in library function of QVT, namely, `rootObjects`, selects the first root object of type `M1` and assigns it to a variable named `m1`, upon which it invokes mapping operation `M1ToN1`.

```
main() {
  var m1 : M1 := M.rootObjects()[M1]->asOrderedSet()->first();
  m1.map M1ToN1();
}
```

A mapping operation includes a signature, a body, optionally, a guard (`when` clause) and a post-condition (`where` clause). QVT also defines several other more complex relationships such as disjunction and inheritance between mapping operations. We deliberately ignore these features in this tutorial for the sake of brevity and also because their effect can be emulated using other more familiar language constructs. Each invocation of a mapping operation results in the instantiation of an instance of the target type of the mapping operation and its subsequent storage in the *extent* of the mapping operation. Model extents annotate instantiation expressions of QVT to explicitly refer to the destination model in which the instantiated object should be stored. Default extents, based on transformation parameters and element types, are assumed if no explicit model extent is give.

The body of a mapping operation consists of three separate sections: initialization, population and end section. The initialization section marked by `init`, which is executed prior to the instantiation of the target model element. Typically, statements in this section define and initialize auxiliary variables and perform intermediate computations that are needed for computing values of attributes of output elements. The population section, which can be optionally marked by `population`, is essentially assignment to the slots of the target instance implicitly instantiated before entering the population and after executing the init section. The instantiated object is referred to by keyword `result`. The following is an example of a mapping operation that maps instances of `M` to `N`, assigning to attribute `att1` of `N1` an attribute of the same name of `M1`. The resulting element is stored in model N as indicated by the model extent annotation.

```
mapping M1::M1ToN1() : N1@N {
    init {
    }
    population {
        result.att1 := self.att1;
        ...
    }
}
```

Mapping operators can be effectively chained for visiting of containment hierarchies of models and transforming each element type to its mapped target as the model is traversed. In our example, `T` maps both elements of type `M3` and `M4` to `N2` while ignoring the intermediate containers of type `M2`. This behavior can readily be denoted in the population section of the `M1ToN1` mapping by invoking two separate mapping operations for types `M3` and `M4`.

```
    ...
    result.n2 := self.m2->m3->flatten()->map M3ToN2()->asOrderedSet();
    result.n2 += self.m2->m4->flatten()->map M4ToN2()->asOrderedSet();
  }
}
```

QVT, like OCL, has two different notations for performing operations on collections and scalar values. Mapping scaler expressions is denoted by dots, and mapping collections is denoted by arrows (i.e., `->`). In the above QVT snippet, `m2` is a container of `M1` elements (containing models of type `M2`), which is a collection in the QVT type system. The first arrow, `->m3`, is technically a syntactic sugar for the `->xcollect(e.m3)` expression, which results in a collection comprising elements of the `m3` container *for each element of* `self.m2`, which is a collection itself. Thus, the result is a collection of collections of type `M3`. In our example, `m1.m2->m3` evaluates to `{{m4},{m5}}`. Built-in operation `flatten`, as the name suggests, removes the nesting, resulting in a flat set of elements of type `M3` containing `{m4, m5}`. The collection is subsequently mapped to a collection of type `N` and assigned to the target element containment feature by invoking the mapping operation and casting its collection type to `OrderedSet`. The second line performs similar operations on type `M4` and merging its result with that of the first mapping invocation. The code for the two mapping operations `M3ToN2` and `M4ToN2` are listed below, respectively.

```
mapping M3::M3ToN2() : N2 {
  att2 := self.att3;
}

mapping M4::M4ToN2() : N2 {
  att2 := self.att4;
}
```

5.3 Synatx of Essential QVT-OM

The abstract syntax of the QVT language are specified using UML in the official OMG specification [10], and is reproduced in Appendix B. For our purpose, we adopt a subset of the language comprising its most essential features, and we opt for a traditional EBNF style for establishing textual abstract syntax necessary for the denotation of other rules later on.

5.3.1 Abstract Syntax

Trans	::=	transformation ID ((Kind ID : Type) ∗) Property ∗ Entry Operation∗
Kind	::=	in | out | inout
Property	::=	property ID Type Exp
Entry	::=	main Block
Operation	::=	Helper | Mapping
Helper	::=	helper ID ((ID : Type) ∗) : Type Block
Mapping	::=	mapping Type :: ID((ID : Type) ∗) : Type Init Population
Init	::=	init Block
Population	::=	population Block
Block	::=	{} | {Exp (; Exp) ∗}
Exp	::=	Literal | Var | Def | Assign | Literal | Type | Call | Map
		| new Type | object Type Exp
		| if Exp then Exp else Exp | while Exp do Exp
		| Exp–>forEach(Var) Block
		| Exp–> (select | collect) (Var|Exp)
		| Block | return Exp | break
Literal	::=	true | false | 0 | 1 . . . | 'string'
	|	(OrderedSet | Set) {Exp∗}
	|	Dict{(Exp = Exp) ∗}
Var	::=	this | self | x | y | · · ·
Call	::=	Exp . (Property | Op(Exp∗))
Map	::=	Exp . map Op(Exp∗)
Def	::=	var Var := Exp
Assign	::=	Exp := Exp
Iter	::=	Exp –> (select | collect) (Var|Exp)

The top-level production in the EBNF grammar for Essential QVT-OM is the Trans rule, which defines a transformation. Most of the excluded features can be readily emulated using the selected ones. Therefore, there is not much loss of expressive power in Essential QVT-OM.

As it was motivated in the previous chapter, to perform effective partial evaluation on model transformations, QvtMix needs to be conscious of the semantic meanings of several built-in library functions, in ordered to correctly glue them with static values. Therefore, we treat a number of the standard OCL library as built-in constructs of the language and provide semantics definitions for them. The chosen library functions are presented in the following.

```
Op ::= =|<>| + | − | ∗ | / |>|<|<=|>=| + =| size | sum | at | indexOf
     | objectsOfType | rootObjects | deepclone | removeElement
     | union | intersection | isEmpty | asSet | asOrderedSet | get | put | hasKey
     | subOrderedSet | exists | forAll | includes | including | excluding
```

5.4 Overall Process and System Architecture

QvtMix is a partial evaluator for the model transformation language QVT-OM. It is, in the full spirit of MDE, a model transformation in its own right, developed and implemented as a QVT-OM program. QvtMix exceedingly leverages the paramount feature of QVT-OM transformations that allow for their treatment as MOF compliant models to create higher-order transformations. It is capable of evaluating and specializing a comprehensive subset of QVT-OM that incorporates its most essential features and has the full expressive power of QVT-OM (the excluded features can be simulated by the chosen one as syntactic sugars).

The partial evaluator is, in essence, a higher order transformation written in QVT; it is implemented as a visitor that uses a certain morphism strategy to evaluate static expressions and mix them with dynamic expressions in the form of residual code. The general algorithm, thus, dispatches these specific transformations for each AST node type. The AST is also represented as models compliant to the QVT metamodel depicted in

Figure B.1, which allows us to denote the partial evaluator in QVT-OM.

QvtMix has a hierarchically heterogeneous architecture underpinning the pipeline depicted in Figure 5.5. The Partial Evaluator pipeline accepts as input a QVT program and a set of static input data, and produces a residual program. Below, we discuss in more detail the major components of QvtMix's architecture:

- **OCL Parser**: This component allows for the parsing and linking of OCL expressions and emits their abstract syntax. As our partial evaluator operates on Ecore elements, the generated abstract syntax is represented as EMF compliant model.

- **QVT Evaluator**: This subsystem realizes a *meta-circular* interpreter for the QVT language–that is, the interpreter is implemented as QVT program itself. The evaluator is capable of wholesome execution of QVT-OM transformations, as well as providing a function called `eval` for the reification of individual OCL and Imperative-OCL expressions. The evaluator stores the values of variables in a lexically scoped environment that is passed to each invocation of the `eval` function. The standard library of the QVT-OM specification is re-exposed (and in a few cases re-implemented). The evaluator is consulted by other components during specialization to evaluate the static values of QVT expressions.

- **Binding Time Analyzer**: QVT uses a hybrid strategy for partial-evaluation; that is, it involves an offline Binding Time Analysis (BTA) phase followed by an online binding time refinement and expression reduction. The main goal of the offline BTA is to determine a tentative binding-time for variables and expressions without evaluating them. It traverses each block of the program and propagates the input division to subsequent expressions. The BTA stores the binding-time of each variable in a binding-time environment. Possible values for binding times are `S` for definitely available values, `D` for definitely dynamic values and `M` for values whose precise binding-time cannot be determined during the offline BTA. The offline BTA is for the most part a value agnostic phase, that is, it does not take into account the actual values of inputs and other variable. It makes a judgement about their binding-times based on the binding-time of other expressions. This is among the reasons for the relatively conservative division that the offline BTA computes. For example, a side-effect free `if` expression both of whose else branch is a static value can be tagged as `S` if the

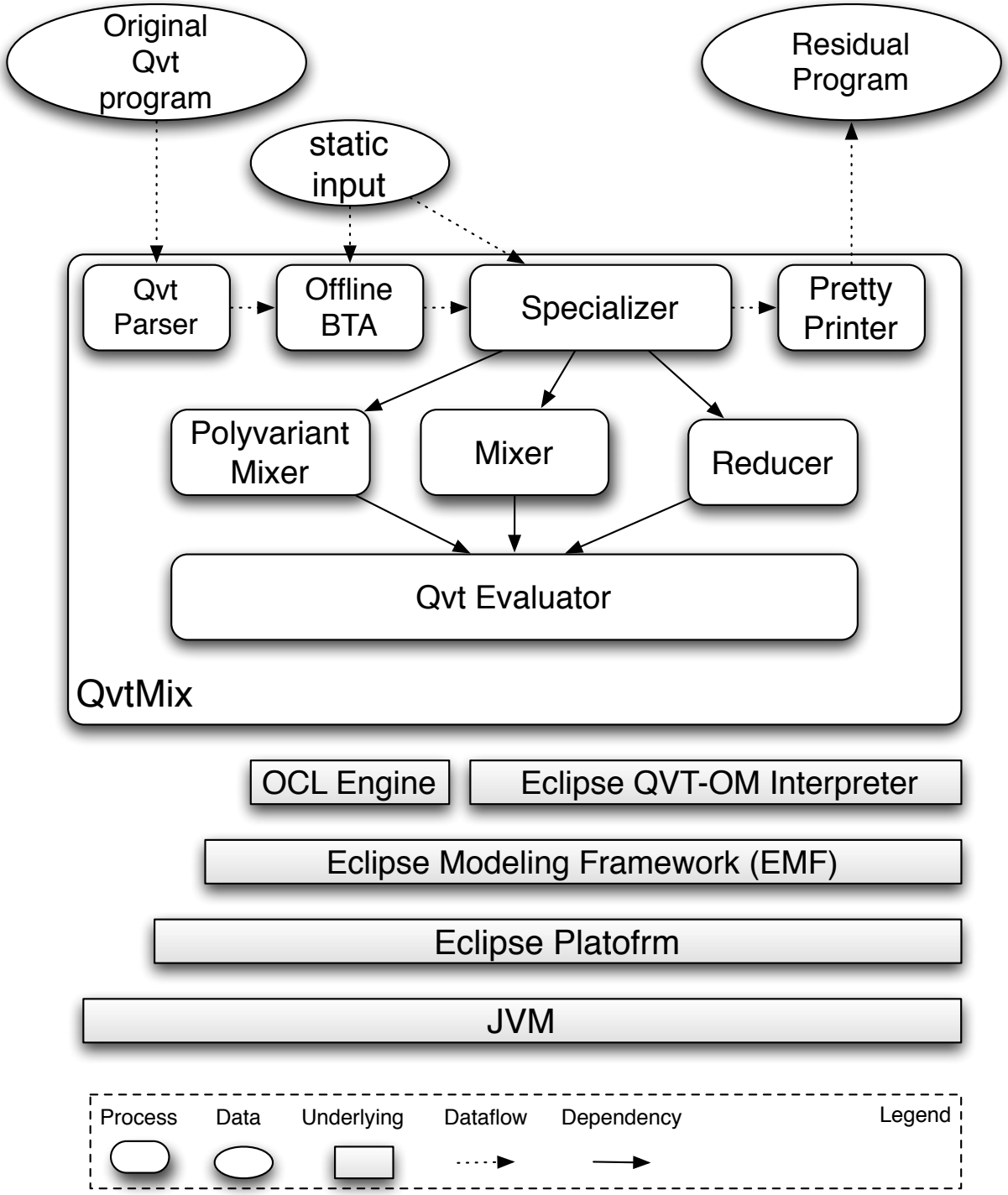


Figure 5.5: System Architecture Diagram

value of its condition expression is statically available (and it is false). Nonetheless, the offline BTA strategy marks it as `M`, for it does not have access to the actual value of the condition expression. To adapt the general partial evaluation process for the case of model transformations, we partition the input model elements into fixed and variable, by annotating their corresponding elements in the input *metamodel* with a change specification. More specifically the change specification designates for each class, container and attribute the possibility of modification (by tagging them as VAR) or lack thereof (by tagging them as FIXED). The BTA propagates the VAR and FIXED tags through the expressions using a special set of inference rules that translates the VAR and FIXED annotations to `S`, `D` and `M` divisions. The specific details of applying partial evaluation on model transformations and models are discussed in more detail in the following subsection.

- **Specializer**: This is the core component of QvtMix. It essentially is a hybrid interpreter/compiler, in the sense that it evaluates the expressions of the program as well as transforms its abstract syntax tree by a number of production rules driven based on the resulting static values. The transformation produces an AST that represents the partially evaluated program. The specializer uses the result of the offline BTA to pinpoint the possibilities of static evaluation. It, however, performs a second online binding-time analysis as it traverses the AST nodes to make final judgement about those expressions whose binding-times could not be resolved during the offline BTA phase. The specializer operates on top of the evaluator and maintains an lexically scope evaluation environment, just like an ordinary interpreter. As such, it can utilize the actual values of static inputs, variables and expressions to recognize more expressions as static, thereby providing opportunities for more aggressive specialization. The result of the online BTA is either `S` or `D`, implying that a final judgement for expressions with uncertain binding-time (i.e., `M`) is due at end of this phase.

  Based on the result of online BTA for each expression, one of the three following strategies are pursued:

  1. **Reducer** for static expressions.
  2. **Mixer** for dynamic expressions in fixed contexts.
  3. **Polyvariant Mixer** for dynamic expressions in variable contexts.

The *reduce* strategy evaluates the result of a static expression, evaluates its side-effects as residual expressions, and returns a list of expressions to replace the value of the expression in the AST and also replicate its side-effects if there is any. On the contrary, the *mix* and *polyvariant mix* strategy is applied on dynamic expressions. These two strategies try to reduce the static subexpressions and *mix* them with the dynamic part. This often involves various *ad hoc* strategies that depend on, among other things, the AST type of the expression, the static type of the values and the values of static subexpressions.

The two types of mixing strategies pertain to the two different kind of contexts that an expression can reside in. The plain (or mono variant) mix is for dynamic expressions that have no context, or have a single, fixed context (e.g, expressions inside the `main` function). Polyvariant mix is, on the other hand, used for dynamic expressions which can be executed in multiple contexts (e.g., inside a body of a mapping operation, which can be invoked on various instances).

- **Pretty Printer**: This component accepts the abstract syntax tree of a QVT-OM transformation and produces source code text executable by the QVT-OM engine.

5.4.1 Partial Evaluation of Model Transformations

The quintessential form of partial evalution as introduced in the previous section requires the subject program to take multiple inputs. A program with a singular input cannot be readily specialized using the scheme laid out in Figure 5.1. The archetypal interface of model transformations appears, however, to be one such program; it takes a single model as input and produce another model as output. This aggregate view is misleading. The truth is models are eclectic collections of multitude of elements, which can be queried and accessed in variety of ways. Therefore, the facade of holism of the model transformation interface does in fact lend itself to partial evaluation. The more important issue to solve is the mechanism to devise a binding-time division, whereby raising the arty of the transformation program.

For the case of model synchronization, there exists a natural division based on the concept of model change in the general synchronization scheme introduced in Chapter 1. Figure 5.6 illustrates how the partial evaluation process is streamlined with respect to this

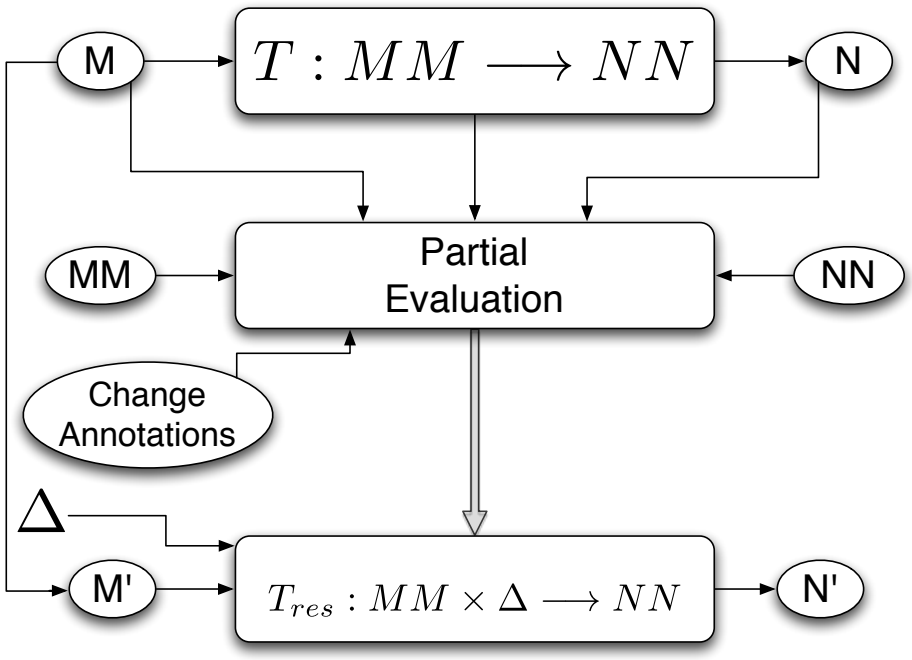


Figure 5.6: Partial Evaluation of Model Transformations

change-driven division. The objective of the process is to provide an equivalent transformation for the given one, that produces the same result on the changed model. Thus, the input model M can, in its entirety, considered as static input. The change specification Δ then constitutes the dynamic part of the model.

To provide better guidelines for the specialization process, the input metamodel are annotated by VAR and FIXED annotations. Along with the transformation, the framework also loads the input and output metamodels, and the change annotation information–that is, the annotations that tell the user which elements of the source models are fixed, and which ones are variable. The BTA framework accepts user annotations of the input metamodel to guide the partial evaluator about the possible classes of changes and locality thereof, according to which the AST of the input transformation is specialized. In the implementation, we use the 'eAnnotation' fragment of Ecore to augment the meta-models with the FIXED and VAR meta-attributes, which are accessible to the QVT-OM partial evaluator.

This explicit annotation is by no means a serious limitation. On the one hand, it is

always possible to provide multiple specialization for the set of applicable changes. This has to do with the fact that the specialization process is performed for each *class* of change on metamodel types (and not each specific change operation on the corresponding model instance). On the other hand, we believe that the implication of this approach for IDEs utilizing model synchronizations based on this technique is congruent with the practice of software development. Just like when programming, the developers' interaction with models adheres to some pattern of locality and temporarily, that is, the most immediate changes are most likely to occur in the most recently modified part of the model and are likely to be of the same type of the most recent modification. In programming, a developer is most likely to continue modifying a class after its inception, by adding several methods to it for example. Likewise, the model developer will likely to add operations to a UML class in a sequential manner. Therefore, the kind of guidance required can be semi-automatically inferred from the user's behavior and the strategy adopted for specialization can be carried out in a tractable fashion.

To facilitate model transformations, QVT-OM and OCL offer various language features to easily manipulate collections in the forms of library functions as well as ingrained language features far beyond what exists in typical general purpose imperative programming languages. These features are widely used for model transformations and provide opportunities for optimization with respect to partially static structures. In particular the general theme for specializing operations that operate on collections is to treat dynamic collections as partially static structures. For the case of ▲ for example, these collections' available elements at specialization time are all considered as the known part of a partially static structure. Those that will be added in the future can be considered to be appended to the collection. The partial evaluator can *memoize* the values of the result of the operation on the static part of the structure and merge it with the dynamic part. The static caching step involves encoding the values of the expression table created in the previous step in the appropriate points in the AST. This is typically in the form of local variables embedded in each context. More specifically, for each variable (or interim value) in a local dictionary variable corresponding to its statically evaluated result is created and populated with the static result of the expressions. Embedding these values sometimes requires delicate manipulation of the AST, to, among other things, ensure the correctness and type safety of the resulting expressions. Also, the expressions need to be transformed into a new form that operate only on the modified parts of the input. This is generally done by utilizing the

information stored in the static cache. The exact details of these operations are dependent on the type of the expression and are discussed formally in Section 5.5.

5.5 Partial Evaluation

QVT-OM has an elaborate syntax posing certain challenges for the task of partial evaluation. To motivate various strategies developed for QvtMix, we present in this section a series of example specialization scenarios that progressively advance towards a typical full-blown model transformation. After each set of examples, we present a formalized specification of the reduction strategy involved in the production of the examples. In each subsection we characterize the steps and intricacies involved in deriving the residual program. We use the following definitions throughout the reset of this chapter.

$$
\begin{array}{rcl}
\mathsf{BT} & ::= & \mathsf{S} \mid \mathsf{D} \mid \mathsf{M} \\
\mathcal{A}\mathit{nnot} & ::= & \mathsf{VAR} \mid \mathsf{VAR}^{\blacktriangle} \mid \mathsf{VAR}^{\blacktriangledown} \mid \mathsf{VAR}^{\blacklozenge} \mid \mathsf{FIXED} \\
\tau & : & \mathcal{E}\mathit{xp} \rightarrow \mathsf{BT} \\
\alpha & : & \mathcal{E}\mathit{xp} \rightarrow \mathcal{A}\mathit{nnot} \\
\Gamma & : & \mathcal{E}\mathit{xp} \rightarrow \mathcal{T}\mathit{ype} \\
\sigma & : & \mathcal{V}\mathit{ar} \rightarrow \mathcal{V}\mathit{al}
\end{array}
$$

In the set of definitions above, BT is the domain of values for binding time of expressions. Static values are designated with S, dynamic ones with D and values binding time of which cannot be definitely determined during the offline phase are designated with M. $\mathcal{A}\mathit{nnot}$ describes the domain of values for annotating metamodel elements to give guidelines about the possible ways in which they can change. VAR is the general annotation for variable elements, which has the three specialized form for each kind of change. Invariant elements are designated with FIXED. Offline BTA is symbolized by binding-time environment τ, which is basically a function from expressions to binding-time values. Similarly, α signifies the annotations of metamodel elements. Expression types are accessed through typing environment Γ. Variable store σ is the evaluation environment that stores the state of the program, and is maintained by the evaluator.

The following operations are similar to boolean disjunctive and conjunctive combinators that aid us to express the binding time inference equations in a concise manner.

$$t_1 \sqcap t_2 = t_2 \sqcap t_1 \qquad D \sqcap D = D \qquad M \sqcap D = D \qquad S \sqcap D = D \qquad S \sqcap M = M \qquad S \sqcap S = S$$

$$t_1 \sqcup t_2 = t_2 \sqcup t_1 \qquad D \sqcup D = D \qquad M \sqcup D = M \qquad S \sqcup D = S \qquad S \sqcup M = S \qquad S \sqcup S = S$$

5.5.1 Offline Binding Time Analysis Rules

The following set of rules specify the most important inference rules used for deducting the binding-time of each expression during the offline BTA phase. Each deduction returns the binding-time of the expression as well as the new BTA. The first definition is a notation simplification that allows us to omit explicitly mentioning τ on the right hand side of the deductions whenever it remains unchanged.

$$\tau \vdash e : t \stackrel{\text{def}}{=} \tau \vdash e : t, \tau$$

$$\frac{\tau \vdash e_1 : t_1, \tau' \qquad \tau' \vdash e_2 : t_2, \tau''}{\tau \vdash e_1; e_2 : t_1 \sqcap t_2, \tau''} \text{ BTA-Seq}$$

$$\frac{e \in \text{Literal}}{\tau \vdash e : S} \text{ BTA-Lit}$$

$$\frac{\tau \vdash e : t}{\tau \vdash \underline{\text{var}}\ x\ :=\ e : t, \tau[x \mapsto t]} \text{ BTA-Init}$$

$$\frac{\tau \vdash e : t}{\tau \vdash x\ :=\ e : t, \tau[x \mapsto t]} \text{ BTA-Var}$$

$$\frac{\tau \vdash e_{cond} : S, e_{then} : t_{then}, e_{else} : t_{else}}{\tau \vdash \underline{\text{if}}\ e_{cond}\ \underline{\text{then}}\ e_{then}\ \underline{\text{else}}\ e_{else} : t_{then} \sqcap t_{else}} \text{ BTA-If-S}$$

$$\frac{\tau \vdash e_{cond} : D \quad (\text{or } M)}{\tau \vdash \underline{\text{if}}\ e_{cond}\ \underline{\text{then}}\ e_{then}\ \underline{\text{else}}\ e_{else} : D \quad (\text{or } M)} \text{ BTA-If-D}$$

$$\frac{\tau \vdash e_{cond} : t_{cond}, e_{body} : t_{do}}{\tau \vdash \underline{\text{while}}\ e_{cond}\ \underline{\text{do}}\ e_{body} : t_{cond} \sqcap t_{do}} \text{ BTA-While-S}$$

$$\frac{\tau \vdash e : t}{\tau \vdash \underline{\text{return}}\ e : t} \text{ BTA-Ret}$$

$$\frac{\alpha \vdash M : \text{VAR}\ \alpha \vdash C : \text{FIXED}\ \Gamma \vdash m : M, c : C \qquad m \in c}{\tau \vdash m : D} \text{ BTA-Model}$$

$$\frac{\alpha \vdash M : a\ \alpha \vdash C : \text{VAR}\ \Gamma \vdash m : M, c : C \qquad m \in c}{\tau \vdash m : D} \text{ BTA-Container}$$

$$\frac{\tau \vdash x : D}{\tau \vdash x.p : D} \text{ BTA-Prop}$$

$$\tau[\![e_s.f(e_1, ..., e_n)]\!] = M \sqcap \Big(\bigsqcap_{i=1 \cdots n} \tau[\![e_i]\!] \Big) \sqcap \tau[\![e_s]\!] \text{ BTA-Call}$$

The first rule, BTA-Seq uses the aforementioned combinator $\sqcap$ to infer the binding-time of a sequence of two statements, which is used repetitively to infer the binding time of blocks of code, e.g., bodies of loop expressions. Constant values obviously have static binding-time, as indicated by rule BTA-Lit. The binding-time of assignment and variable

definition expressions are the same as that of their right hand-side expressions (rules BTA-Init and BTA-Var). These two rules modify the BT A environment τ by adding a mapping from the variable name to its associated binding-time. If the binding-time of the condition of an If expression is dynamic, then the whole expression is recognized as dynamic (BTA-If-D). However, if the condition is static, the binding-time of the If expression is inferred to be the conjunction of its *then* and else expressions' binding-time. Rules BTA-Model and BTA-Container link the binding-time value to the annotation values; the former states that the binding-time of a model element $\mathfrak{m}$ which is an instance of meta-model type M is dynamic, if M is annotated as Var, whereas the latter assigns $\mathfrak{m}$ to D if its container has VAR annotation. Finally, the binding-time of call expressions depend on that of the source of the call and each argument of the call, as indicated by the BTA-Call rule.

5.5.2 Online Binding Time Analysis Rules

$$\mathcal{B}[\![]\!] : \mathsf{Exp} \times \mathsf{Env} \times (\mathsf{Exp} \to \mathsf{BT}) \longrightarrow \{S, D\} \qquad \mathcal{B}[\![e; e']\!] = \mathcal{B}[\![e]\!] \sqcap \mathcal{B}[\![e']\!]$$

$$\mathcal{B}_{\mathsf{Exp}}[\![e]\!] = \begin{cases} S & \tau \vdash e : S \\ D & \text{otherwise} \end{cases} \qquad \mathcal{B}_{\mathsf{Var}}[\![x]\!] = \begin{cases} D \sqcup \tau[\![x]\!] & \sigma(x) = \bot \\ S \sqcup \tau[\![x]\!] & \text{otherwise} \end{cases}$$

$$\mathcal{B}[\![\underline{\mathsf{var}}\ \mathsf{x}\ \mathtt{:=}\ e]\!] = \mathcal{B}[\![e]\!] \qquad \mathcal{B}[\![\mathsf{x}\ \mathtt{:=}\ e]\!] = \mathcal{B}[\![e]\!]$$

$$\mathcal{B}[\![\underline{\mathsf{if}}\ e_{cond}\ \underline{\mathsf{then}}\ e_{then}\ \underline{\mathsf{else}}\ e_{else}]\!] =$$

$$\begin{cases} \begin{cases} \mathcal{B}[\![e_{then}]\!] & \mathcal{E}[\![e_{cond}]\!] = \mathsf{True} \\ \mathcal{B}[\![e_{else}]\!] & \mathcal{E}[\![e_{cond}]\!] = \mathsf{False} \end{cases} & \mathcal{B}[\![e_{cond}]\!] = S \\ \begin{cases} S & \mathcal{E}[\![e_{then}]\!] = \mathcal{E}[\![e_{else}]\!] \\ D & \text{otherwise} \end{cases} & \mathcal{B}[\![e_{else}]\!] \sqcap \mathcal{B}[\![e_{then}]\!] = S \\ D & \text{otherwise} \end{cases} \quad \text{otherwise}$$

$$\text{let } (T_s, T_r, \langle (a_i, T_i) \rangle, e_{body}) \leftarrow \zeta(f) \quad \mathcal{B}[\![e_s\mathtt{.f(}e_1, .., e_n\mathtt{)}]\!] =$$

$$\begin{cases} \tau[a_i \mapsto \mathcal{B}[\![e_i]\!]]\mathcal{B}[\![e_{body}]\!] & \tau[\![e_s\mathtt{.f(}e_1, .., e_n\mathtt{)}]\!] = M \\ \tau[\![e_s\mathtt{.f(}e_1, .., e_n\mathtt{)}]\!] & \text{otherwise} \end{cases}$$

As explained in Section 5.4, QvtMix uses a hybrid approach for the final determination of binding-time values of expressions. The offline phase's rules were denoted in the previous subsection. The above set of rules specify the online BTA phase. Function $\mathcal{B}[\![.]\!]$ computes the online binding-time of its argument based on an environment and according to the offline binding time information passed to it. A notable rule is the one used for refining the binding-time of If expressions. Because the static values are now available to the partial evaluator, the value of the condition expression can be consulted and a decision based on that be made. If the binding time of the condition is static and its value is True, the binding-time of the If expression is equivalent to that of its then-condition. Also, if the binding time o the condition is dynamic, but both branches are static and evaluate to the same value, the binding time of the If-expression is static.

5.5.3 Constant Propagation and Static Expression Evaluation

The first example demonstrate partial evaluation for expressions whose operands are constant or static. The variables holding the results of these expressions are also determined to be static. Consequently, the succeeding expressions dereferencing these variables are also statically pre-evaluated. In the following example variable `x` is initialized with an expression consisting of two constant sub-expression. The offline BTA evaluates each operand as static. Therefore, the initialization expression is also tagged as static, and so is the variable `x`. The binding time for the variable is then stored in the binding-time environment which the offline BTA maintains. Proceeding to the definition of variable `y`, the offline BTA resolves operand `2` as static, and fetches from the binding-time environment its prior resolution of the binding-time of the variable `x` (which was also static); hence variable `y`'s binding time is also determined as static and the binding-time assignment is added to the environment.

```
1 main() {
2   var x := 1 + 1;
3   var y := x + 2;
4   var z := y * 2;
5   var t := x + z;
6 }
```

QvtMix ⇉

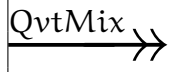
```
1 main() {
2     var x : Integer := 2;
3     var y : Integer := 4;
4     var z : Integer := 8;
5     var t : Integer := 10;
6 }
```

The *reduce* strategy is employed for rewriting all of the expressions above. Due to the fact that the expressions are all side-effect free, the statically evaluated values are simply in lieu of the original initialization expressions. It is intuitively evident that the two pieces of code above behave exactly the same, but the specialized one on the left performs considerably fewer number of computations than the one on the right.

5.5.4 Loops

Loop bodies are assessed for the existence of side-effects. If the side-effects of the body of a loop only affect the value of some variables and those values depend only on static expressions, then the loop is replaced by the reduce strategy with a number of assignments to variables whose values are modified as a result of executing the loop. This is exemplified in the following piece of code.

```
1  main() {
2    var i := 1;
3    var s := 0;
4    var p := 1;
5    while (i < 10) {
6      s := s + i;
7      p := p * i;
8      i := i + 1;
9    };
10 }
```




```
1  main() {
2      var i : Integer := 1;
3      var s : Integer := 0;
4      var p : Integer := 1;
5      s := 45;
6      p := 362880;
7      i := 10;
8  }
```

However, if the loop body depends on a dynamic value, or if it has side-effects, the mixReduce strategy is invoked. The following example remains unchanged due to variable `size` having dynamic binding time.

```
1  main() {
2    var x := bm.objectsOfType(Book)->asOrderedSet();
3    var size := x->size();
4    var i := 1;
5    var s := 0;
6    while (i < 10) {
7      s := s + i * size;
8      i := i + 1;
9    };
10 }
```

The following piece of code also remains unchanged due to the side effect of the log expression.

```
1  main() {
2    var i := 1;
3    var s := 0;
4    while (i < 10) {
5      s := s + i;
6      log('',s);
7      i := i + 1;
8    };
9  }
```

5.5.5 Conditional Expressions

```
1 main() {
2   if 2 > 1 then
3     log('true')
4   else
5     log('false')
6   endif;
7 }
```

QvtMix

```
1 main() {
2     log('true');
3 }
```

The following piece of cod however remains unchanged because the condition expression is cannot be statically reduced.

```
1 main() {
2   var books := bm.objectsOfType(Book)->asOrderedSet();
3   if books->size() > 0 then
4     log('true')
5   else
6     log('false')
7   endif;
8 }
```

In the following example, even though the condition expression is dynamic, both branches have d side-effect free expressions. QvtMix eliminates the if-expression, replacing it by the value of the branches. Note that, QvtMix is able to recognize side-effects; if each branch were a `log` statement, albeit identical, the if-expression would remain intact.

```
1 main() {
2   var books := bm.objectsOfType(
        Book)->asOrderedSet();
3   var x := if books->size() > 0
        then
4     1 + 1
5   else
6     2
7   endif;
8 }
```

QvtMix

```
1 main() {
2     var books : OrderedSet(Book)
          := bm.objectsOfType(Book)->
          asOrderedSet();
3     var x : Integer := 2;
4 }
```

5.5.6 Symbolic Expression Simplification

The partial evaluator performs various forms of algebraic manipulation of expressions, so as to simplify the generated code and also eliminate local redundant sub-expressions. The following snippets, illustrate this:

```
1 main() {
2   var books :=  bm.objectsOfType(
        Book)->asOrderedSet();
3   var x := books->size();
4   var y := (2 * x + 1) + (3 * x -
        4);
5 }
```

QvtMix

```
1 main() {
2     var books : OrderedSet(Book)
          := bm.objectsOfType(Book)->
          asOrderedSet();
3     var x : Integer := books->size
          ();
4     var y : Integer := 5 * x - 3;
5 }
```

5.5.7 Reduce Rules

$$\mathcal{R}[\![]\!] : \mathsf{Exp} \times \mathsf{Env} \longrightarrow \langle \mathsf{Exp} \rangle \times \mathsf{Env}$$

$$\mathcal{SE}[\![]\!] : \mathsf{Exp} \times \mathsf{Env} \longrightarrow \langle \mathsf{Exp} \rangle \times \mathsf{Env}$$

$$[\![]\!] : \mathsf{Val} \longrightarrow \mathsf{Exp}$$

$$\mathcal{R}_{\mathsf{Exp}}[\![e]\!]\sigma = (\langle e \rangle, \sigma) \qquad \mathcal{R}[\![e; e']\!] = \mathcal{R}[\![e]\!] \oplus \mathcal{R}[\![e']\!]$$

$$\textit{where} \begin{cases} \oplus : \langle \mathsf{Exp} \rangle \times \mathsf{Env} \times (\mathsf{Env} \to \langle \mathsf{Exp} \rangle \times \mathsf{Env}) \longrightarrow \langle \mathsf{Exp} \rangle \times \mathsf{Env} \\ (c, \sigma) \oplus f = (c \uplus \pi_1(f(\sigma)), \pi_2(f(\sigma))) \end{cases}$$

$$\mathcal{R}[\![\underline{\mathsf{while}}\ e_{cond}\ \underline{\mathsf{do}}\ e_{body}]\!] = \mathcal{SE}[\![e_{body}]\!]$$

$$\mathcal{R}[\![\underline{\mathsf{if}}\ e_{cond}\ \underline{\mathsf{then}}\ e_{then}\ \underline{\mathsf{else}}\ e_{else}]\!] = \begin{cases} \mathcal{R}[\![e_{then}]\!] & \mathcal{E}[\![e_{cond}]\!] = \mathsf{True} \\ \mathcal{R}[\![e_{else}]\!] & \mathcal{E}[\![e_{cond}]\!] = \mathsf{False} \end{cases}$$

$$\mathcal{R}[\![\underline{\mathsf{var}}\ x\ \mathtt{:=}\ e]\!]\sigma = (\mathcal{SE}[\![e]\!] \uplus \langle \underline{\mathsf{var}}\ x\ \mathtt{:=}\ [\![\mathcal{E}[\![e]\!]]\!] \rangle, \sigma[x \mapsto \mathcal{E}[\![e]\!]])$$

$$\mathcal{R}[\![x\ \mathtt{:=}\ e]\!]\sigma = (\mathcal{SE}[\![e]\!] \uplus \langle x\ \mathtt{:=}\ [\![\mathcal{E}[\![e]\!]]\!] \rangle, \sigma[x \mapsto \mathcal{E}[\![e]\!]])$$

$$\mathcal{R}[\![\underline{\mathsf{return}}\ e]\!] = \mathcal{SE}[\![e]\!] \uplus \langle \underline{\mathsf{return}}\ [\![\mathcal{E}[\![e]\!]]\!] \rangle$$

The example introduced in this subsections are all reduced using a collection of reduction rules, some of which are presented above. The reduce function, $\mathcal{R}[\![.]\!]$, takes as input an expression and an environment, and returns a sequence of expression re-writing the original one along with a modified environment. In the rules above, the side-effect evaluation function, $\mathcal{SE}[\![.]\!]$, is a function that besides evaluating expressions, also computes their possible side-effects and returns a list of expressions that mimic those side-effects, e.g., assignment to modified variables in the reduction of while loops as discussed earlier.

The rule for reducing sequence of statements uses a special operator that denotes the plumbing required to pass modified store from one statement to another, and also amalgamate the results of the reduction of each expression with those of its predecessor. Its definition basically states that the returning expression list is the concatenation of the expression list given as its first argument with what its second argument, a function, cal-

culates. The returning environment is the result of applying the second argument to the passed environment (often processed by the earlier statement). The signature for this operator is devised so as to incorporate that of the reduce function as its arguments.

The reduction of variable initialization and assignment, as expected, modify the environment, mapping the variable referenced by the expression to the value of the right hand side of the expression. Conditional expressions are reduced to one of their branches depending on the value of their condition expression.

5.5.8 Contexts and Function Calls

The examples in this section exemplify several of the strategies used for partially evaluating function calls to ordinary functions (i.e., helper functions in QVT-OM). Due to the special semantics and peculiarities associated with mapping calls, they are discussed in a separate section. The first example is series of call to a unary pure function that calculates the square of its argument. QvtMix is able to evaluate the function with the static argument and substitute the function call with it. As the example demonstrates, QvtMix properly handles the case of calls with variables (with static binding times) as well as nested calls.

```
1 main() {
2   var x := sqr(2);
3   var y := sqr(x);
4   var z := sqr(sqr(x));
5 }
6 helper sqr(i : Integer) : Integer
    {
7   return i * i;
8 }
```

QvtMix ⟫

```
1 main() {
2     var x : Integer := 4;
3     var y : Integer := 16;
4     var z : Integer := 256;
5 }
```

Calls to recursive functions with no side-effects with arguments having static binding time are also similarly reduced to their statically evaluated values.

```
1  main() {
2    var y := fact(3);
3  }

5  helper fact(n : Integer) : Integer
       {
6    if n = 1 then {
7      return 1;
8    }
9    endif;
10   return n * fact(n - 1);
11 }
```

QvtMix ⟫

```
1  main() {
2      var y : Integer := 6;
3  }
```

In general, function calls cannot be always reduced to a singular value. The following example deals with the case when a binary function is invoked with a static and a dynamic argument. In this case, QvtMix specializes the called function, namely `mul`, with respect to the static value. The specialized function is a unary function that only takes the dynamic value, and whose constituting expressions are reduced according to the static value bound to its first argument (i.e., `i` is bound to two). Finally, a new call to the specialized function is inserted in the calling expression in lieu of the original call.

```
1  main() {
2    var b := bm.objectsOfType(Book)
         ->asOrderedSet();
3    var x := mul(b->size(), 2);
4  }
5  helper mul(i : Integer, j :
       Integer) : Integer {
6    return i * j;
7  }
```

QvtMix ⟫

```
1  main() {
2    var b : OrderedSet(Book) := bm.
         objectsOfType(Book)->
         asOrderedSet();
3    var x : Integer := this.
         sqr__mix1(b->size());
4  }
5  helper sqr__mix1(i : Integer) :  {
6    return i * 2;
7  }
```

If the called function has side-effects then the call is not reduced. The following thus example remains intact.

```
main() {
  var x := hello();
}
helper hello() : Integer{
  log('hello ');
  return 0;
}
```

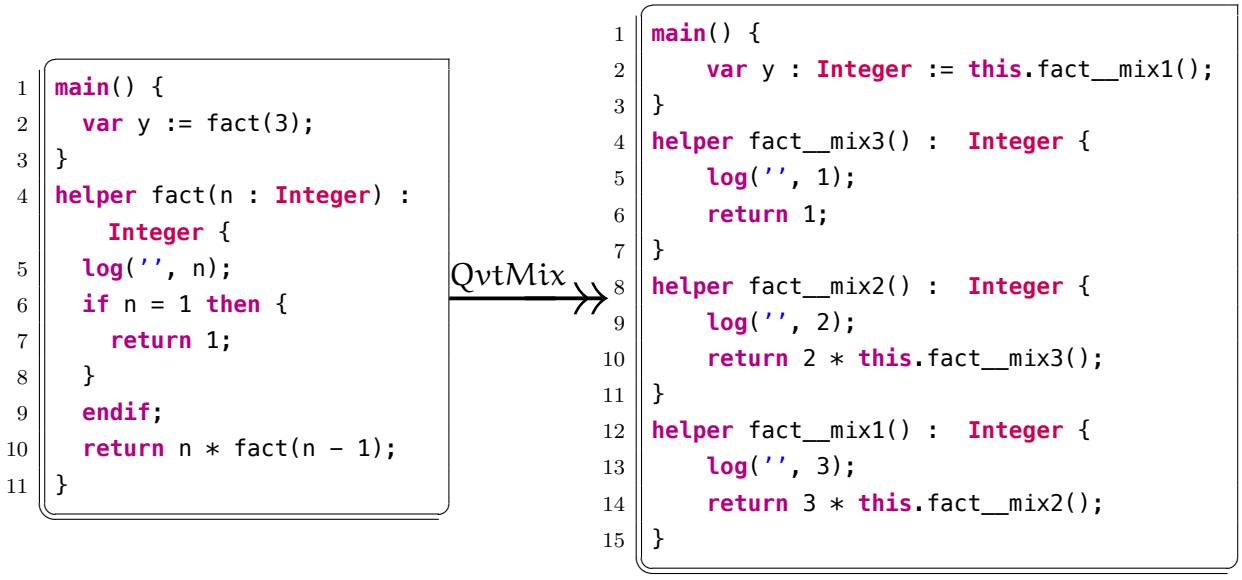
If the function has side-effects and multiple arguments, then partial reduction is performed. In the following example, the `hello` function has a statement with side-effect (i.e., `log`) and is called with one static argument, viz., `s`. A specialized version of this function is created that inlines the static value.

```
main() {
  var x := hello('world');
}
helper hello(s : String) :
    Integer{
  log('hello ' + s);
  return i;
}
```

QvtMix ⇉

```
main() {
 var x : Integer := this.hello__mix1();
}
helper hello__mix1() : Integer {
 log('hello world');
 return i;
}
helper hello(s : String) : Integer{
 log('hello ' + s);
 return i;
}
```

A peculiar case is the partial evaluation of recursive functions that have side-effects. QvtMix is able to reduce such calls by creating a specialized function for each iteration of the call as it descends through the recursive calls of the function and chaining them together to produce the same computational and side effects. In the following example, the specialization of the call to the `fact` function yields three different specializations cascaded to one-another to emulate the original recursive chain.



5.5.9 Reducing Iterate Expressions

As discussed in Chapter 5.2, QVT-OM and OCL offer a number of collection iteration constructs aiming to facilitate the inquiry and manipulation of model element containers. More specifically, `collect` and `select` (and their imperative counterparts `xcollect` and `xselect`) provide succinct mechanisms to apply respectively mapping or query operations on collections. QvtMix is capable of partially evaluating these constructs under a variety of conditions. The first example illustrates the effect of partial evaluation when the collection over which the iterate expression is applied. On the left hand-side, variable `xs` is assigned to a static value of type `OrderedSet(Integer)`. The next statement is a `xselect` query that runs over `xs`, returning a collection comprising those elements of `xs` that are greater than two. This expression can hence be fully evaluated at specialization-time. In contrast, the first part of the last statement–that is, `b.chapters->xcollect(nbPages)`–creates a collection from a dynamic source expression (i.e., `b.chapters`). As such, it can only be evaluated at compile time. The argument of the `union` function call in the second part of the last statement, however, only depends on `ys`, a static variable. It, therefore, is fully reduced to a static value and is inlined in place of the argument of the `union` function call. The example assumes that the annotations of the input meta-model are somehow that the expression `b.chapters->collect(nbPages)` is determined to be fully-dynamic (e.g, property `nbPages`

is declared as VAR).

```
1 main() {
2   var b := bm.objectsOfType
        (Book)->asOrderedSet()
        ->first();
3   var xs := OrderedSet
        {1,2,3,4};
4   var ys := xs->xselect(x|x
         > 2);
5   var zs := b.chapters->
        collect(nbPages)->
        union(ys->xcollect(y|y
        *y));
6 }
```

QvtMix

```
1 main() {
2     var b : Book := bm.objectsOfType(Book
          )->asOrderedSet()->first();
3     var xs : OrderedSet(Integer) :=
          OrderedSet{1, 2, 3, 4};
4     var ys : OrderedSet(Integer) :=
          OrderedSet{3, 4};
5     var zs : Sequence(Integer) := b.
          chapters->collect(temp1 : Chapter|
          temp1.nbPages)->union(Sequence{9,
          16});
6 }
```

5.5.10 Mixed Reduction of Partially Static Collections

Insertion of new elements into containers of model elements is effectively tantamount to appending new values at the end of collections in the domain of QVT-OM expressions. Therefore, several strategies of mix reduction of dynamic collections deal with the recurring pattern where collection related operations are performed over a sequence that can have extra elements in the future. Given an ordered collection such as C, if we can somehow divide it into two sub-sequence of old and new elements, then for certain operations we can reuse the result of the previous run of the transformation over the old part, perform the operation on the new elements and finally merge the two, thereby saving redundant computations that would otherwise be performed processing the old elements. Figure 5.7 summarizes the steps required to reduce a collection iterate expression under a fixed context.

In the following example, assume that:

$$\begin{aligned} \mathtt{b} = \; & (\langle\langle(\varnothing, \langle\text{"ch1"}, \text{"20"}\rangle, \varnothing, \mathtt{Chapter}), (\varnothing, \langle\text{"ch2"}, \text{"10"}\rangle, \varnothing, \mathtt{Chapter})\rangle\rangle, \\ & \langle\text{"lib1"}, \text{"book1"}\rangle, \varnothing, \mathtt{Book}) \end{aligned}$$

.

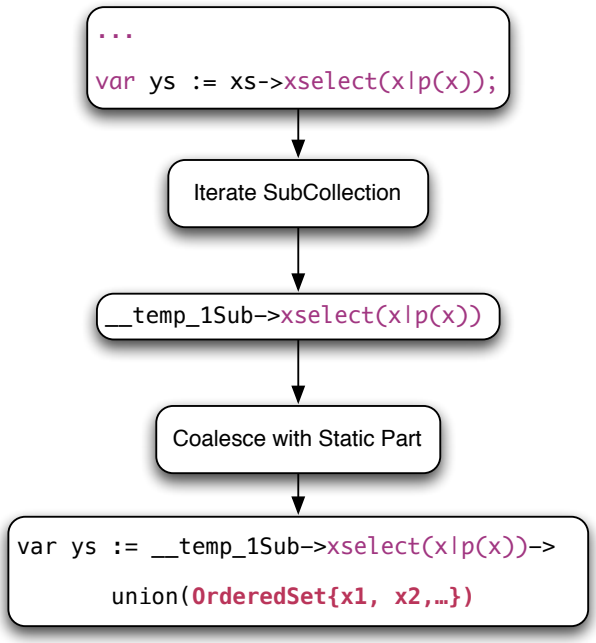


Figure 5.7: Monovariant Mix Reduction

```
1  main() {
2   var b := bm.
        objectsOfType(
        Book)->
        asOrderedSet()->
        first();

4   var zs := b.
        chapters->
        xcollect(nbPages
        );
5  }
```

QvtMix

```
1  main() {
2      var b : Book := bm.objectsOfType(Book)->
           asOrderedSet()->first();
3      var __temp_1 := b.chapters;
4      var __temp_1Last := 2;
5      var __temp_1Size := __temp_1->size();
6      var __temp_1Sub :=
7        if (__temp_1Size > __temp_1Last) then
8          __temp_1->subOrderedSet(__temp_1Last +
               1, __temp_1Size)
9        else
10         OrderedSet{}
11       endif;

13     var zsSub := __temp_1Sub->xcollect(temp1 :
           Chapter|temp1.nbPages);

15     var zs : Sequence(Integer) := zsSub->union(
           Sequence{20, 10}->asBag());
16 }
```

The right hand side piece of code has three auxiliary variables:

1. `__temp_1Last`: a static variable which holds the position of the last static element of the collection
2. `__temp_1Size`: a variable assigned to the dynamic size of the collection
3. `__temp_1Sub`: the dynamic sub-collection of the source

Variable `__temp_1Sub` extracts the dynamic part of this partially static structure, i.e., those elements that are added to the model after the partial evaluation of the transformation. They are available during the execution time. This separation is done by the aid of the `__temp_1Last` variable, which holds the size of the collection at the time of partial evaluation.

The last two lines is where the operation is performed and merged with the static result. First, the `xcollect` operation is invoked on the dynamic fragment of the collection stored in `__temp_1Sub`, the result of which is stored in variable `zSub`. The initialization expression of the original variable (i.e., `z`) is assigned to an expression which is the merge of `zSub` with the result of the static evaluation of the expression on the existing collection, which for our example is `Sequence{20,10}`. The merge of these two collections are done using the `union` function.

There are often cases where embedding the result of the static evaluation into the host expression requires adjustment to the type of the computed expression to satisfy the QVT-OM type-checker. The example listed above is one such case, and as it demonstrates, QvtMix is able to properly augment residual expressions with the required glueing type-cast operations.

5.5.11 Mix Rules

$$\mathcal{M}[\![]\!] : \mathsf{Exp} \times \mathsf{Env} \longrightarrow \mathsf{Exp} \times \langle \mathsf{Exp} \rangle \times \mathsf{Env}$$

$$\mathcal{M}_{\mathsf{Exp}}[\![e]\!] = (e, \varnothing) \qquad \mathcal{M}[\![e; e']\!] = \mathcal{M}[\![e]\!] \oplus \mathcal{M}[\![e']\!]$$

$$\text{let } (T_s, T_r, \langle (a_i, T_t) \rangle, e_{body}) \leftarrow \zeta(f)$$

$$\mathcal{M}[\![e_s\texttt{.f(}e_1, .., e_n\texttt{)}]\!] = \begin{cases} (\pi_1(\mathcal{M}[\![e_s]\!])\texttt{.f(}e_1, .., e_n\texttt{)}, \pi_2(\mathcal{M}[\![e_s]\!])) & bt = S \\ (\pi_1(\mathcal{M}[\![e_s]\!])\texttt{.}[\![f_{mix}]\!]\texttt{(}e'_1, .., e'_n\texttt{)}, E) & \text{otherwise} \end{cases}$$

$$\mathcal{M}[\![e_s\texttt{.}\underline{\text{map}}\ \texttt{f(}e_1, .., e_n\texttt{)}]\!] = \begin{cases} (\pi_1(\mathcal{M}[\![e_s]\!])\texttt{.}\underline{\text{map}}\ \texttt{f(}e_1, .., e_n\texttt{)}, \pi_2(\mathcal{M}[\![e_s]\!])) & bt = S \\ (\pi_1(\mathcal{M}[\![e_s]\!])\texttt{.}\underline{\text{map}}\ [\![f_{mix}]\!]\texttt{(}e'_1, .., e'_n\texttt{)}, E) & \text{otherwise} \end{cases}$$

$$\begin{aligned} \textit{where}\quad & bt = \mathcal{B}[\![e_s]\!] \sqcap (\bigsqcap_{i=1}^{n} \mathcal{B}[\![e_i]\!]) \\ & \tau' = \tau[a_i \mapsto \mathcal{B}[\![e_i]\!]|_{i=1..n}, \texttt{self} \mapsto \mathcal{B}[\![e_s]\!]] \\ & \sigma' = \sigma[a_i \mapsto \mathcal{E}[\![e_i]\!]|_{\mathcal{B}[\![e_i]\!]=S}, \texttt{self} \mapsto \mathcal{E}[\![es]\!]] \\ & f_{mix} = \mathcal{PE}[\![f]\!]\sigma'\tau' \\ & e'_i = \begin{cases} \pi_1(\mathcal{M}[\![e_i]\!]) & \mathcal{B}[\![e_i]\!] = D \\ \mathcal{E}[\![e_i]\!] & \mathcal{B}[\![e_i]\!] = S \end{cases} \\ & E = \pi_2(\mathcal{M}[\![e_s]\!]) \uplus \biguplus_{\mathcal{B}[\![e_i]\!]=D} \pi_2(\mathcal{M}[\![e_i]\!]) \end{aligned}$$

$$\mathcal{M}[\![\underline{\text{var}}\ \texttt{x := } e]\!] = (\underline{\text{var}}\ \texttt{x := } e_{mix}, E_{mix})$$

$$\mathcal{M}[\![\texttt{x := } e]\!] = (\texttt{x := } e_{mix}, E_{mix})$$

$$\textit{where}\quad (e_{mix}, E_{mix}) = \mathcal{M}[\![e]\!] \curlyvee \mathcal{E}[\![e]\!]$$

$$\mathcal{M}[\![\underline{\text{if}}\ e_{cond}\ \underline{\text{then}}\ e_{then}\ \underline{\text{else}}\ e_{else}]\!] = \begin{cases} \begin{cases} \mathcal{M}[\![e_{then}]\!] & \mathcal{E}[\![e_{cond}]\!] = \mathsf{True} \\ \mathcal{M}[\![e_{else}]\!] & \mathcal{E}[\![e_{cond}]\!] = \mathsf{False} \end{cases} & \mathcal{B}[\![e_{cond}]\!] = S \\ (\underline{\text{if}}\ e_{cond}\ \underline{\text{then}}\ e_{then}\ \underline{\text{else}}\ e_{else}, \varnothing) & \text{otherwise} \end{cases}$$

$$\mathcal{M}[\![\underline{\text{return}}\ e]\!] = (\underline{\text{return}}\ \pi_1(\mathcal{M}[\![e]\!]), \pi_2(\mathcal{M}[\![e]\!]))$$

The rules for mix reduction strategies used by QvtMix are presented above. Similar

to the reduce function, the mix function takes an expression and an environment. It, however, returns a surrogate expression, directly replacing the reduced one, as well as a list of auxiliary expressions that are inserted before the expression being reduced. These auxiliary expressions are often variables used to store intermediate results, or glue code for, among other things, satisfying the type-checker. It also returns a modified environment capturing changes made to variables to enable the correct static evaluation of succeeding expressions that reference those variables.

The same combinator used for the reduce strategy is also used here for combining the results and the effects of sequences of expressions. Of particular importance are the two rules pertaining to calls to helper functions and mapping operations. In both cases, a specialized function called $\mathtt{f}_{\mathtt{mix}}$ is created by applying the partial evaluator to an environment that corresponds to the context of the function. The static arguments are evaluated and bound to their corresponding parameters. Variable `self` is also bound to the value of the source expression. The partial evaluator produces a version of the function that takes only the arguments whose binding-time are dynamic. The expressions in the body of the function are mix-reduced based on the static values bound to static parameters.

For assignment and variable initialization expressions, QvtMix uses the merge operator (denoted in the rules by $\curlyvee$) for coalescing the static sub-expressions with the residual ones. Merge is a polymorphic operator that dispatches different strategies for merging based on the type of its operands. Several interesting cases of merging are presented in the following.

$$\curlyvee\colon (\mathsf{Exp} \times \langle \mathsf{Exp} \rangle) \times \mathsf{Val} \times \mathsf{Env} \longrightarrow \mathsf{Exp} \times \langle \mathsf{Exp} \rangle \qquad (e, \mathsf{E}) \curlyvee \nu = (e, \mathsf{E})$$

$$(e_s\texttt{->sum()}, \mathsf{E}) \curlyvee \nu = (\llbracket e_s\texttt{->sum()} + \llbracket \nu \rrbracket \rrbracket, \mathsf{E})$$

$$(e_s\texttt{->}\underline{\texttt{select}}\texttt{(x|}e_p\texttt{)}, \mathsf{E}) \curlyvee \nu = (e_s\texttt{->}\underline{\texttt{select}}\texttt{(x|}e_p\texttt{)->union(}\llbracket \nu \rrbracket\texttt{)}, \mathsf{E})$$

$$(e_s\texttt{->}\underline{\texttt{collect}}\texttt{(x|}e_c\texttt{)}, \mathsf{E}) \curlyvee \nu = (e_s\texttt{->}\underline{\texttt{collect}}\texttt{(x|}e_c\texttt{)->union(}\llbracket \nu \rrbracket\texttt{)}, \mathsf{E})$$

The following two rules elaborate the discussed methodology used for the mix reduction of `select` and `collect` expressions.

$$
\begin{array}{l}
\mathcal{M}[\![e_s\texttt{->}\underline{\texttt{select}}\texttt{(x|}e_p\texttt{)}]\!] = \\
\quad (\\
\quad \texttt{u->}\underline{\texttt{select}}\texttt{(x|}e_p\texttt{)}, \\
\quad \pi_2(\mathcal{M}[\![e_s]\!]) \uplus \\
\quad \langle \\
\quad\quad \underline{\texttt{var}}\ \texttt{s} := \pi_1(\mathcal{M}[\![e_s]\!]), \\
\quad\quad \underline{\texttt{var}}\ \texttt{l} := |\mathcal{E}[\![e_s]\!]|, \\
\quad\quad \underline{\texttt{var}}\ \texttt{z} := \texttt{s.size()}, \\
\quad\quad \underline{\texttt{var}}\ \texttt{u} := \underline{\texttt{if}}\ \texttt{s} > \texttt{l}\ \underline{\texttt{then}}\ \texttt{s->subOrderedSet(l} + 1, \texttt{z)}\ \underline{\texttt{else}}\ \underline{\texttt{OrderedSet}}\texttt{\{\}} \\
\quad \rangle \\
\quad) \\
\\
\mathcal{M}[\![e_s\texttt{->}\underline{\texttt{collect}}\texttt{(x|}e_c\texttt{)}]\!] = \\
\quad (\\
\quad \texttt{u->}\underline{\texttt{collect}}\texttt{(x|}e_c\texttt{)->union(}e_s\texttt{->subOrderedSet(}1, \texttt{l)->}\underline{\texttt{collect}}\texttt{(x|}\pi_1(\mathcal{M}[\![e_c]\!])\texttt{))}, \\
\quad \pi_2(\mathcal{M}[\![e_s]\!]) \uplus \\
\quad \langle \\
\quad\quad \underline{\texttt{var}}\ \texttt{s} := \pi_1(\mathcal{M}[\![e_s]\!]), \\
\quad\quad \underline{\texttt{var}}\ \texttt{l} := |\mathcal{E}[\![e_s]\!]|, \\
\quad\quad \underline{\texttt{var}}\ \texttt{z} := \texttt{s.size()}, \\
\quad\quad \underline{\texttt{var}}\ \texttt{u} := \underline{\texttt{if}}\ \texttt{s} > \texttt{l}\ \underline{\texttt{then}}\ \texttt{s->subOrderedSet(l} + 1, \texttt{z)}\ \underline{\texttt{else}}\ \underline{\texttt{OrderedSet}}\texttt{\{\}} \\
\quad \rangle \uplus \\
\quad \biguplus_{\nu_i \in \mathcal{E}[\![e s]\!]} \pi_2(\mathcal{M}[\![e_c]\!]\sigma[x \mapsto \nu_i]) \\
\quad)
\end{array}
$$

These two rules are based on the fact that these two lemmas for combining the result of these operations on new values in a collection with those obtained from the previous evaluation of the expression on the old collection.

Lemma 5.5.1

$$\mathcal{E}[\![\mathsf{c}_1 \uplus \mathsf{c}_2\text{->}\underline{\mathsf{select}}(x \mid e_p)]\!] = \mathcal{E}[\![\mathsf{c}_1\text{->}\underline{\mathsf{select}}(x \mid e_p)]\!] \uplus \mathcal{E}[\![\mathsf{c}_2\text{->}\underline{\mathsf{select}}(x \mid e_p)]\!]$$

and

$$\mathcal{E}[\![\mathsf{c}_1 \uplus \mathsf{c}_2\text{->}\underline{\mathsf{collect}}(x \mid e_c)]\!]\sigma = \mathcal{E}[\![\mathsf{c}_1\text{->}\underline{\mathsf{collect}}(x \mid e_c)]\!]\sigma \uplus \mathcal{E}[\![\mathsf{c}_2\text{->}\underline{\mathsf{collect}}(x \mid e_p)]\!]\sigma$$

Proof. Straightforward induction on c_2 using the $\mathsf{E}-\mathit{select}$ and $\mathsf{E}-\mathit{select}-\mathtt{Empty}$ big-step operational semantics rules presented in Appendix C. Base case $\mathsf{c}_2;\sigma \Downarrow \langle \nu_1 \rangle$. Induction step: if for $\mathsf{c}_2;\sigma \Downarrow \langle \nu_1, .., \nu_n \rangle$ the lemma holds, then it also holds for $\mathsf{c}_2;\sigma \Downarrow \langle \nu_1, .., \nu_n, \nu_{n+1} \rangle$. The proof of the second equation is similar to this one.

□

In the residual code emanated for mixed reduction of `select` and `collect`, there are a number auxiliary variables. More specifically, variable $\mathtt{s}$ is the mix of the source expression, variable $\mathtt{l}$ holds the position of the static part of the collection and variable $\mathtt{z}$ holds the dynamic size. Variable $\mathtt{u}$ is the new source for the operation call, which picks the dynamic part of the collection, i.e., the elements appended at the end of the static collection of size $\mathtt{l}$.

For the case of `select` expression, the query is run over the dynamic fragment of collection (referenced as $\mathtt{u}$). In contrast, the reduced code for `collect` applies the mixed version of the collect expression on the static portion of the collection, which should be separated at runtime using the call to `subOrderedset` library function. The reason is that, the `collect` expressions can be used for applying, among other things, mapping operations on collections. A residual version of these operations should still be applied on the static part of the collection. This has to do with the need to support the situations where imperative mapping operations may access values from model elements other than those they are directly applied on. The transformation scenario presented in Section 5.6 exemplifies such a situation. In both cases, when the expression is assigned to a variable (or used to initialize one) the residue expression is merged with the literal value obtained from the static evaluation of the expression during the time of partial evaluation (see previous rules for variable assignment and initialization).

5.5.12 Partial Evaluation of Mapping Operations

Mapping operations are the chief constructs of QVT-OM for model transformations. The call to a chain of mapping operation is distinguished from calling helper functions by the property that the returning call effectively instantiates a new object and serializes the returning objects to the output model. A mapping operation is a of a variable context and is typically applied on model elements of different values. It is not practical to produce a specialized version of the mapping operation for each single model element in the input. Thus, the mapping operations need to be specialized in a way that they can correctly transform the existing model elements as well as the new ones. The general theme of polyvariat-mix strategies employed for mapping operations is to place the static values for each invocation of the mapping operation in a static tables rather than inline them as the vanilla mix does.

For the examples in this section, consider the metamodels depicted in Figure 5.8. The input meta-model on the right hand side is a domain model containing a simple containment hierarchy. The containing class, `Book`, has two attributes and one container. The contained types, `Chapter`, has also two properties: `nbPages` and `title` that denote the number of pages and the title of the chapter, respectively. The metamodel on the right-hand side contains the type `Publication` that is an abstraction of the two classes on the left.

The following transformation creates a `Publication` class for each `Book` instance. It assigns the `name` attribute of the created `Publication` instance to the `title` attribute of the corresponding `Book`, and aggregates the `nbPages` of all chapters of the book into the `nbPages` attribute of the target model element, i.e., `Chapter` instance.

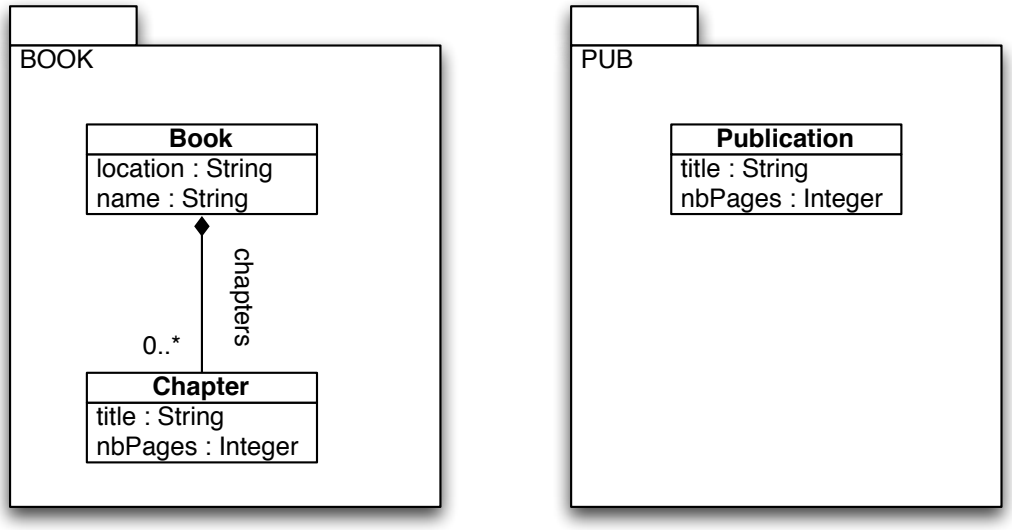


Figure 5.8: PUB and BOOK Metamodels

```
1  transformation Book2Pub(in bm : BOOK, out lm : PUB);
2
3  main() {
4    var b := bm.objectsOfType(Book)->asOrderedSet()->last();
5    var x := b.map M();
6  }
7  mapping Book::M() : Publication {
8    title := self.name.toUpper();
9    nbPages := self.chapters->xcollect(nbPages)->sum();
10 }
```

The mapping operation M is an example of a mapping that does not create a one-to-one correspondence between the elements of the source and target. It is in fact what is referred to in the technical literature of function programming as *catamorphism*[12]. These are the type of mappings that collapse the structure of their input into a smaller, aggregate unit, thereby loosing some information from the input but projecting a summary representation thereof. The white-box approach is particularly apt for successfully handling these kinds of transformations, as other approaches, including the black-box synchronization scheme introduced in Chapter 4, have several shortcomings in establishing traces for non-monotonic transformations.

The transformation above exemplifies some of the common characteristics of model transformations specified in such hybrid languages as QVT-OM. In this language, primarily due to its OCL heritage, several transformation rules operate on collections of model elements that are selected based on context-dependent criteria. The set of input elements in the source model is divided into a set of FIXED elements whose values and relationships are known and will not change, and variable elements, labeled as VAR. Such information about model elements are annotated in the source metamodel. Consider the model transformation above to be applied routinely on a source model to which progressively more elements of type `Chapter` are added. The metamodel annotations for this change is presented in Figure 5.9.

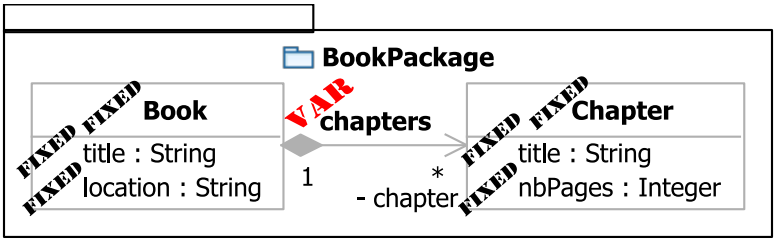
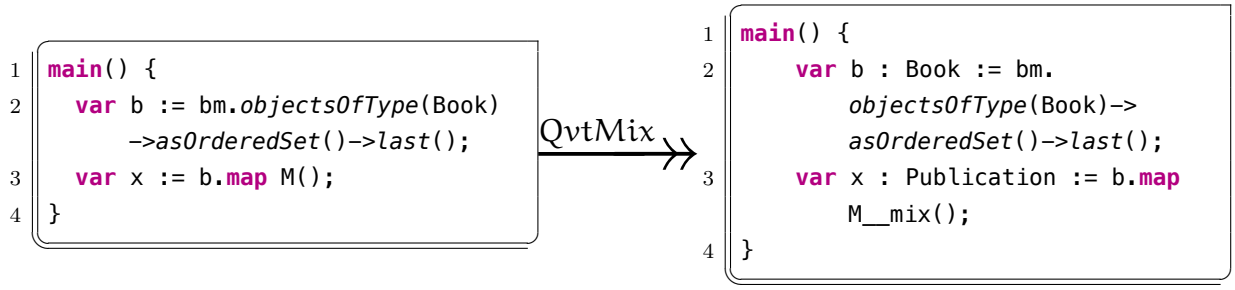


Figure 5.9: Annotations for static analysis

This particular annotation labels `Book` class as FIXED, which indicates that neither any new instances of this class will be added, nor will any of the existing ones be removed from the input model. In contrast, `chapters` composition association is labeled as VAR to declare that the future iterations of this model will have new instances of `Chapter`(For simplicity, we only consider addition here). All attributes of classes have FIXED annotations which means their values are invariant.

The entry operation of `Book2Pub` transformation, i.e., `main`, is transformed according to the mix reduction strategy. When the mapping operation call on line 5 is encountered, QvtMix creates a specialization of the called mapping, to wit, `M__mix` and creates a call expression with the same source to the new mapping:



The body of the mapping is specialized according to the polyvariant mix reduce strategy. What distinguishes this strategy from the simple mix strategy is its caching of different contextual intermediate values in global static tables rather than embedding them in the code. For mapping `M` three such global tables are created. Each table is basically a dictionary that maps the context of the mapping operation to the value of the cached variable. The first cached value pertains to the first statement in `M` on line 8. The computed static value of `title` is held in a dictionary (line 3 below). The two other caches correspond to the `xcollect` statement in the mapping operation. As discussed before, one keeps track of the position of static values in the collection. The third one caches the static result computed for variable `nbPages`, i.e., the value of the built-in operation `sum`.

The listing in the following presents the mixed mapping operation. The statement on line 7 sets the value of the variable that keeps the last position of the static part of the `self.chapters` collection, which is read from the corresponding table by the `get` operation. Since the archetype of contexts of mapping operations are model elements and they may be large structures, they are quite possibly not very efficient if directly used as keys for hashing the static values in tables. Instead, the address of the model elements in the input models are used as keys for the dictionaries; hence the use of `path` operation on line 7 to get the path of the object in the model.

```
1 property __M__mix__result_nbPagesCache : Dict(String, Integer) = Dict { '/allBooks.4' =
    70 };
2 property __M__mix__temp_1LastCache : Dict(String, Integer) = Dict { '/allBooks.4' = 2 };
3 property __M__mix__result_titleCache : Dict(String, String) = Dict { '/allBooks.4' = '
    ODYSSEY' };
```

The *VAR* annotation on `chapters` container entails that the value of `Chapter::nbPages` attribute depends on the chapters that will be added later on. However, by specializing this

transformation for addition of input elements, the partial evaluator can infer the result for the existing chapters. In line 11, the result of the `sum` operation performed on the dynamic fragment of `self.chapters` is merged with the static value which is similarly fetched from the first cache.

```
1  mapping Book::M__mix() : Publication
2  {
3      object result : Publication@lm {
4          result.title := self.name.toUpper();
5
6          var __M__mix__temp_1 := self.chapters;
7          var __M__mix__temp_1Last := __M__mix__temp_1LastCache->get(self.oclAsType(EObject
               ).path());
8          var __M__mix__temp_1Size := __M__mix__temp_1->size();
9          var __M__mix__temp_1Sub := if (__M__mix__temp_1Size > __M__mix__temp_1Last) then
               __M__mix__temp_1->subOrderedSet(__M__mix__temp_1Last + 1,
               __M__mix__temp_1Size)   else OrderedSet{}   endif;
10         var __result_nbPagesSub := __M__mix__temp_1Sub->xcollect(temp1 : Chapter|temp1.
               nbPages)->sum();
11         result.nbPages := __result_nbPagesSub + __M__mix__result_nbPagesCache->get(self.
               oclAsType(EObject).path());
12     };
13 }
```

5.5.13 Polyvariant Mix Rules

This section presents the formal definition of the polyvariant reduction rules, used for the partial evaluation of expressions in that can be executed in multiple contexts, e.g., those in the body of mapping operations. Similar to other reduction rules discussed earlier, the polyvariant mix reduce function, $\mathcal{PM}[\![.]\!]$, takes an expression as its argument and returns a primary replacement expression, along with a list of auxiliary expressions.

$$\mathcal{PM}[\![]\!] : \mathsf{Exp} \times \mathsf{Context} \times \mathsf{Env} \times \Psi \longrightarrow \mathsf{Exp} \times \langle \mathsf{Exp} \rangle \times \mathsf{Env} \times \Psi$$

$$\Psi = \mathsf{Context} \times \mathsf{Var} \longrightarrow (\mathsf{Addr} \times \mathsf{Val}) \qquad \mathcal{PM}_{\mathsf{Exp}}[\![e]\!] = \mathcal{M}[\![e]\!]$$

$$\mathcal{PM}[\![e; e']\!] = \mathcal{PM}[\![e]\!] \oplus \mathcal{PM}[\![e']\!]$$

$$\mathcal{PM}[\![\underline{\mathsf{var}}\ \mathsf{x}\ \texttt{:=}\ e]\!]\ \mathsf{c}\ \sigma\ \psi = (\underline{\mathsf{var}}\ \mathsf{x}\ \texttt{:=}\ e_{mix}, \mathsf{E}_{mix}, \sigma'', \psi[\mathsf{c} :: \mathsf{x} \mapsto (\mathsf{Addr}_{\mathcal{M}}(\sigma(\mathsf{self})), \mathcal{E}[\![e]\!])])$$

$$\mathcal{PM}[\![\mathsf{x}\ \texttt{:=}\ e]\!]\ \mathsf{c}\ \sigma\ \psi = (\mathsf{x}\ \texttt{:=}\ e_{mix}, \mathsf{E}_{mix}, \sigma'', \psi[\mathsf{c} :: \mathsf{x} \mapsto (\mathsf{Addr}_{\mathcal{M}}(\sigma(\mathsf{self})), \mathcal{E}[\![e]\!])])$$

where

$$\sigma' = \sigma[\mathsf{x} \mapsto \mathcal{E}[\![e]\!]]$$
$$\mathsf{l} = [\![\psi(\mathsf{c} :: \mathsf{x})]\!]\texttt{->get(}\mathsf{Addr}_{\mathcal{M}}(\sigma(\mathsf{self}))\texttt{)}$$
$$(e', \mathsf{E}') = \mathcal{M}[\![e]\!] \curlyvee \mathsf{l}$$

$$(e_{mix}, \mathsf{E}_{mix}, \sigma'') = \begin{cases} \begin{cases} (\mathsf{u}\texttt{->f(}e_1, .., e_n\texttt{)}, \\ \mathsf{E}' \uplus \langle \underline{\mathsf{var}}\ \mathsf{u}\ \texttt{:=}\ e_s \rangle, & \sigma(\mathsf{u}) = \bot \\ \sigma'[\mathsf{u} \mapsto \mathcal{E}[\![e_s]\!]) \end{cases} & \\ & e \neq e' = e_s\texttt{->f(}e_1, .., e_n\texttt{)} \\ \begin{cases} (\mathsf{u}\texttt{->f(}e_1, .., e_n\texttt{)}, \\ \mathsf{E}' \uplus \langle \mathsf{u}\ \texttt{:=}\ e_s \rangle, & \text{otherwise} \\ \sigma'[\mathsf{u} \mapsto \mathcal{E}[\![e_s]\!]) \end{cases} & \\ (e', \mathsf{E}', \sigma') & \text{otherwise} \end{cases}$$

Besides the usual variable store environment, the polyvariant mix function takes two additional argument. The first one signifies the context in which the expression is being reduced. The second one is the collection of the static caches created by QvtMix to memoize the intermediate results of expressions for each context. This table, represented

by ψ, associates to each variable and context a static cache that maps the address of each model element to the value that the variable assumes when the context is executed over said model element. More specifically, if the mapping operation `Context::op()` is invoked over a collection of model elements (e.g., `OrderedSet{m1, m2, m3}->map op()`), then each cached result in the body of `op` is a mapping from the addresses of `m1`, `m2` and `m3` to the value of the expression when `op` is invoked on the respective element.

The rules for polyvariant reduction of assignment and variable initialization expressions are essentially the same as their respective mix reduction rules with two important difference. A static cache is added to hold the static value for current context. For expressions other than calls, a new initialization expression is formed by merging the mix reduced version of the original initialization expression with a lookup expression that fetches the contextual static value from the cache (represented by $\mathfrak{l}$). For call expressions, an auxiliary variable, $\mathfrak{u}$, is also created, which is initialized to the static value of the source of the call expression. The original source expression for the call, viz. e_s, is replaced by this variable.

5.5.14 Polyvariant Reduction of Iterate Expressions

The following two rules present the polyvariant mix reduction of collection iterate expressions. Again, the most prominent difference with the vanilla mix rules are the use of static caches, in lieu of inline literals, to represent static values. In both cases, the variable holding the position of the last static element, namely $\mathfrak{l}$, is initialized with a lookup expression to read the value for the context from its pertinent cache. The modified Ψ states caching of the static value of $\mathfrak{l}$.

$$
\begin{array}{l}
\mathcal{PM}[\![e_s\texttt{->}\underline{\texttt{select}}\texttt{(x|}e_p\texttt{)}]\!]\ c\ \sigma\ \psi = \\
\quad (\\
\quad u\texttt{->}\underline{\texttt{select}}\texttt{(x|}e_p\texttt{)}, \\
\quad \pi_2(\mathcal{PM}[\![e_s]\!]) \uplus \\
\quad \langle \\
\qquad \underline{\texttt{var}}\ \texttt{s := } \pi_1(\mathcal{PM}[\![e_s]\!]), \\
\qquad \underline{\texttt{var}}\ \texttt{l := } [\![\psi(c :: x)]\!]\texttt{->get(}\mathrm{Addr}_M(\sigma(self))\texttt{)}, \\
\qquad \underline{\texttt{var}}\ \texttt{z := s.size()}, \\
\qquad \underline{\texttt{var}}\ \texttt{u := } \underline{\texttt{if}}\ \texttt{s} > \texttt{l}\ \underline{\texttt{then}}\ \texttt{s->subOrderedSet(l} + 1, \texttt{z)}\ \underline{\texttt{else}}\ \underline{\texttt{OrderedSet}}\texttt{\{\}} \\
\quad \rangle, \\
\quad \sigma, \\
\quad \psi[c :: l \mapsto (\mathrm{Addr}_M(\sigma(self)), |\mathcal{E}[\![e_s]\!]|)] \\
\quad) \\
\\
\mathcal{PM}[\![e_s\texttt{->}\underline{\texttt{collect}}\texttt{(x|}e_c\texttt{)}]\!]\ c\ \sigma\ \psi = \\
\quad (\\
\quad u\texttt{->}\underline{\texttt{collect}}\texttt{(x|}e_c\texttt{)->union(}e_s\texttt{->subOrderedSet(}1, \texttt{l)->}\underline{\texttt{collect}}\texttt{(x|}\pi_1(\mathcal{M}[\![e_c]\!])\texttt{))}, \\
\quad \pi_2(\mathcal{PM}[\![e_s]\!]) \uplus \\
\quad \langle \\
\qquad \underline{\texttt{var}}\ \texttt{s := } \pi_1(\mathcal{PM}[\![e_s]\!]), \\
\qquad \underline{\texttt{var}}\ \texttt{l := } [\![\psi(c :: x)]\!]\texttt{->get(}\mathrm{Addr}_M(\sigma(self))\texttt{)}, \\
\qquad \underline{\texttt{var}}\ \texttt{z := s.size()}, \\
\qquad \underline{\texttt{var}}\ \texttt{u := } \underline{\texttt{if}}\ \texttt{s} > \texttt{l}\ \underline{\texttt{then}}\ \texttt{s->subOrderedSet(l} + 1, \texttt{z)}\ \underline{\texttt{else}}\ \underline{\texttt{OrderedSet}}\texttt{\{\}} \\
\quad \rangle \uplus \\
\quad \biguplus_{v_i \in \mathcal{E}[\![es]\!]} \pi_2(\mathcal{M}[\![e_c]\!]\sigma[x \mapsto v_i]), \\
\quad \sigma, \\
\quad \psi[c :: l \mapsto (\mathrm{Addr}_M(\sigma(self)), |\mathcal{E}[\![e_s]\!]|)] \\
\quad)
\end{array}
$$

5.6 A Full Example

To further introduce the capabilities QvtMix we extend the previous example with a few other subtleties that require the expressive power of QVT-OM to be effectively specified. The following is the listing of the transformation. Lines 11, 12 and 13 are the difference between this transformation and the previous example. What these do is to query the source model for other books with the same name, and then make the `title` of the `Publication` instance to the form of `<bookName>_<i>_of_<n>`, where `n` represents the total number of books with the same name in the source model, and `i` is the index of this particular book in that list.

```
1  modeltype PUB  uses 'http://pub/1.0';
2  modeltype bookModel uses 'http://book/1.0';
3  transformation Book2Pub(in bm : bookModel, out lm : PUB);

5  main() {
6    var books := bm.objectsOfType(Book)->asOrderedSet();
7    var x := books->map Book2Pub();
8  }

10 mapping Book::Book2Pub() : Publication {
11   var books := bm.objectsOfType(Book)->asOrderedSet()->xselect(b|b.name = self.name);
12   var index := books->indexOf(self);
13   title := self.name + '__' + index.toString() + '_of_' + books->size().toString();
14   nbPages := self.chapters->xcollect(nbPages)->sum();
15 }
```

The full code of the partially evaluated transformation specialized for the input model of Figure 5.10 is reproduced below:

QVT Code

```
1  import qvt.mix.util;
2  modeltype ECORE uses ecore('http://www.eclipse.org/emf/2002/Ecore');
3  modeltype PUB uses pub('http://pub/1.0');
4  modeltype bookModel uses book('http://book/1.0');

6  transformation Book2Pub(in bm: bookModel, out lm: PUB);
```

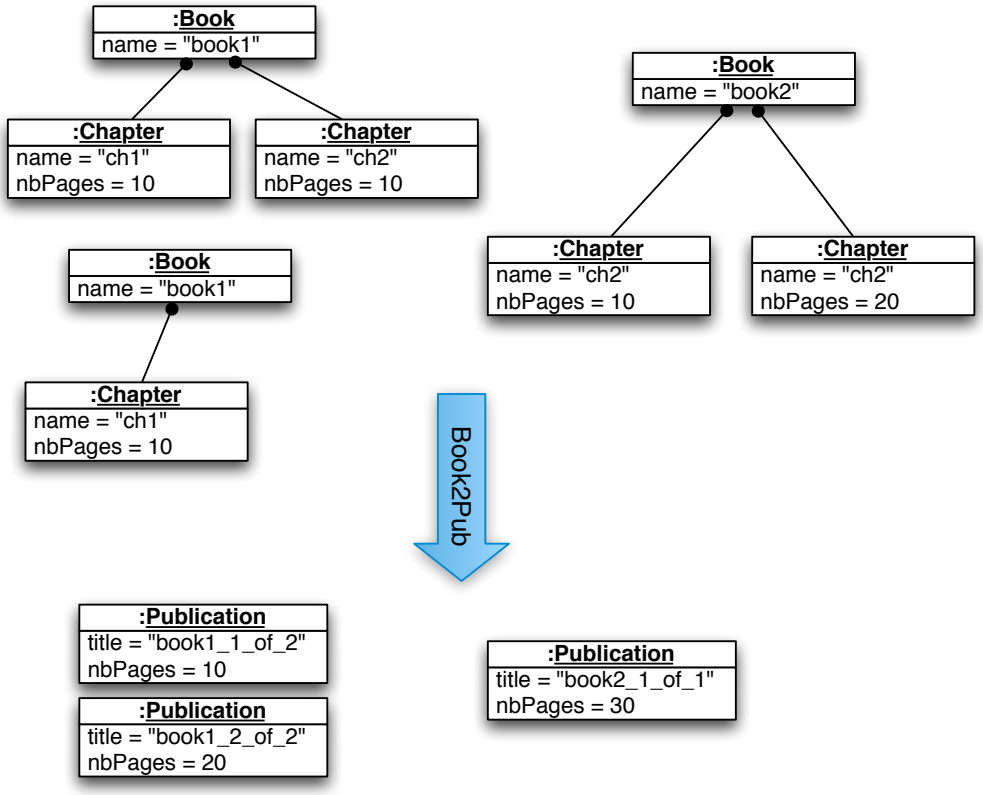


Figure 5.10: The Source and Target Instance Models

```
property __Book2Pub__mix__result_nbPagesCache : Dict(String, Integer) = Dict { '/allBooks.2' = 20, '/allBooks.3' = 10, '/
    allBooks.1' = 20 };
property __Book2Pub__mix__booksCache : Dict(String, OrderedSet(ObjectPath)) = Dict { '/allBooks.2' = OrderedSet{object
    ObjectPath {
    path := '/allBooks.2';
}}, '/allBooks.1' = OrderedSet{object ObjectPath {
    path := '/allBooks.1';
}, object ObjectPath {
    path := '/allBooks.3';
}}, '/allBooks.3' = OrderedSet{object ObjectPath {
    path := '/allBooks.1';
}, object ObjectPath {
    path := '/allBooks.3';
}} };
property __Book2Pub__mix__indexCache : Dict(String, Integer) = Dict { '/allBooks.3' = 2, '/allBooks.2' = 1, '/allBooks.1'
     = 1 };
property __Book2Pub__mix__temp_2LastCache : Dict(String, Integer) = Dict { '/allBooks.3' = 1, '/allBooks.1' = 2, '/
    allBooks.2' = 2 };
property __Book2Pub__mix__result_titleCache : Dict(String, String) = Dict { '/allBooks.2' = 'book2__1_of_1', '/allBooks.3
    ' = 'book1__2_of_2', '/allBooks.1' = 'book1__1_of_2' };
property __Book2Pub__mix__temp_1LastCache : Dict(String, Integer) = Dict { '/allBooks.2' = 3, '/allBooks.3' = 3, '/
    allBooks.1' = 3 };
```

```
main() {
    var books : OrderedSet(Book) := bm.objectsOfType(Book)->asOrderedSet();
    var __temp_1 := books;
    var __temp_1Last := 3;
    var __temp_1Size := __temp_1->size();
    var __temp_1Sub := if (__temp_1Size > __temp_1Last) then __temp_1->subOrderedSet(__temp_1Last + 1, __temp_1Size)
         else OrderedSet{}  endif;
    var x : Sequence(Publication) := __temp_1Sub->xcollect(temp1 : Book|temp1.map Book2Pub())->union(books->subOrderedSet
         (1, __temp_1Last)->xcollect(temp1 : Book|temp1.map Book2Pub__mix()));
}

mapping Book::Book2Pub() : Publication
{
    object result : Publication@lm {
        var books : OrderedSet(Book) := bm.objectsOfType(Book)->asOrderedSet()->xselect(b : Book|b.name = self.name);
        var index : Integer := books->indexOf(self);
        result.title := self.name + '__' + index.toString() + '_of_' + books->size().toString();
        result.nbPages := self.chapters->xcollect(temp1 : Chapter|temp1.nbPages)->sum();
    };
}

mapping Book::Book2Pub__mix() : Publication
{
    object result : Publication@lm {
        var __Book2Pub__mix__temp_1 := bm.objectsOfType(Book)->asOrderedSet();
        var __Book2Pub__mix__temp_1Last := __Book2Pub__mix__temp_1LastCache->get(self.oclAsType(EObject).path());
        var __Book2Pub__mix__temp_1Size := __Book2Pub__mix__temp_1->size();
        var __Book2Pub__mix__temp_1Sub := if (__Book2Pub__mix__temp_1Size > __Book2Pub__mix__temp_1Last) then
             __Book2Pub__mix__temp_1->subOrderedSet(__Book2Pub__mix__temp_1Last + 1, __Book2Pub__mix__temp_1Size)   else
             OrderedSet{}   endif;
        var __booksSub := __Book2Pub__mix__temp_1Sub->xselect(b : Book|b.name = self.name);
        var books := __booksSub->union(__Book2Pub__mix__booksCache->get(self.oclAsType(EObject).path())->getObject(bm)->
             oclAsType(Book)->asBag());
        var __indexSub := __Book2Pub__mix__indexCache;
        var index := __indexSub->get(self.oclAsType(EObject).path());
        result.title := self.name + '__' + index.toString() + '_of_' + books->size().toString();
        var __Book2Pub__mix__temp_2 := self.chapters;
        var __Book2Pub__mix__temp_2Last := __Book2Pub__mix__temp_2LastCache->get(self.oclAsType(EObject).path());
        var __Book2Pub__mix__temp_2Size := __Book2Pub__mix__temp_2->size();
        var __Book2Pub__mix__temp_2Sub := if (__Book2Pub__mix__temp_2Size > __Book2Pub__mix__temp_2Last) then
             __Book2Pub__mix__temp_2->subOrderedSet(__Book2Pub__mix__temp_2Last + 1, __Book2Pub__mix__temp_2Size)   else
             OrderedSet{}   endif;
        var __result_nbPagesSub := __Book2Pub__mix__temp_2Sub->xcollect(temp1 : Chapter|temp1.nbPages)->sum();
        result.nbPages := __result_nbPagesSub + __Book2Pub__mix__result_nbPagesCache->get(self.oclAsType(EObject).path())
             ;
    };
}


////////////////////////// ModelUtils ////////////////////////////////////////////////////
intermediate class ObjectPath {
  path : String;
}

helper EObject::path() : String {
  var parents : OrderedSet(Integer) := OrderedSet{};
  var containers : OrderedSet(Integer) := OrderedSet{};
  var path := '';
  var obj := self;
  var parentObj := obj.eContainer();
  while (parentObj <> null) {
```

```
81      var container := obj.eContainingFeature().oclAsType(EReference);
82      var col := Sequence{};
83      getMultiFeature(parentObj, container.name, col);
84      var position := col->indexOf(obj);
85      path := '/' + container.name + '.' + position.repr() + path;
86      obj := parentObj;
87      parentObj := obj.eContainer();
88    };
89    if path = '' then {
90      path := '/';
91    }
92    endif;
93    return path;
94  }

96  helper Model::getObject(p : ObjectPath) : EObject {
97    var str := p.path;
98    var obj := self.rootObjects()->asOrderedSet()->first().oclAsType(EObject);
99    str := str.substringAfter('/');
100   while (str <> null) {
101     var segment := str.substringBefore('/');
102     var pos : String;
103     var cont : String;
104     if segment <> null then {
105       pos := segment.substringAfter('.');
106       cont := segment.substringBefore('.');
107     }
108     else {
109       pos := str.substringAfter('.');
110       cont := str.substringBefore('.');
111     }
112     endif;
113     obj := obj.getObject(cont, pos.asInteger());
114     str := str.substringAfter('/');
115   };

117   return obj;
118 }

120 helper ObjectPath::getObject(model : Model) : EObject {
121   return model.getObject(self);
122 }

124 helper EObject::getObject(cont : String, pos : Integer) : EObject {
125   var col := Sequence{};
126   getMultiFeature(self, cont, col);
127   return col->at(pos).oclAsType(EObject);
128 }
```

From Line 69 onward, there are addenda helper functions that implement the path scheme used for addressing objects inside models and storing values in static tables. As these functionalities are missing fro the standard library of QVT, the code for them are patched to the end of each specialized transformation that requires them. Now if we add a new book with name `book1` to the source model, both original and specialized transformations create an instance of `Publication` on the target model. Figure 5.11 exemplifies a

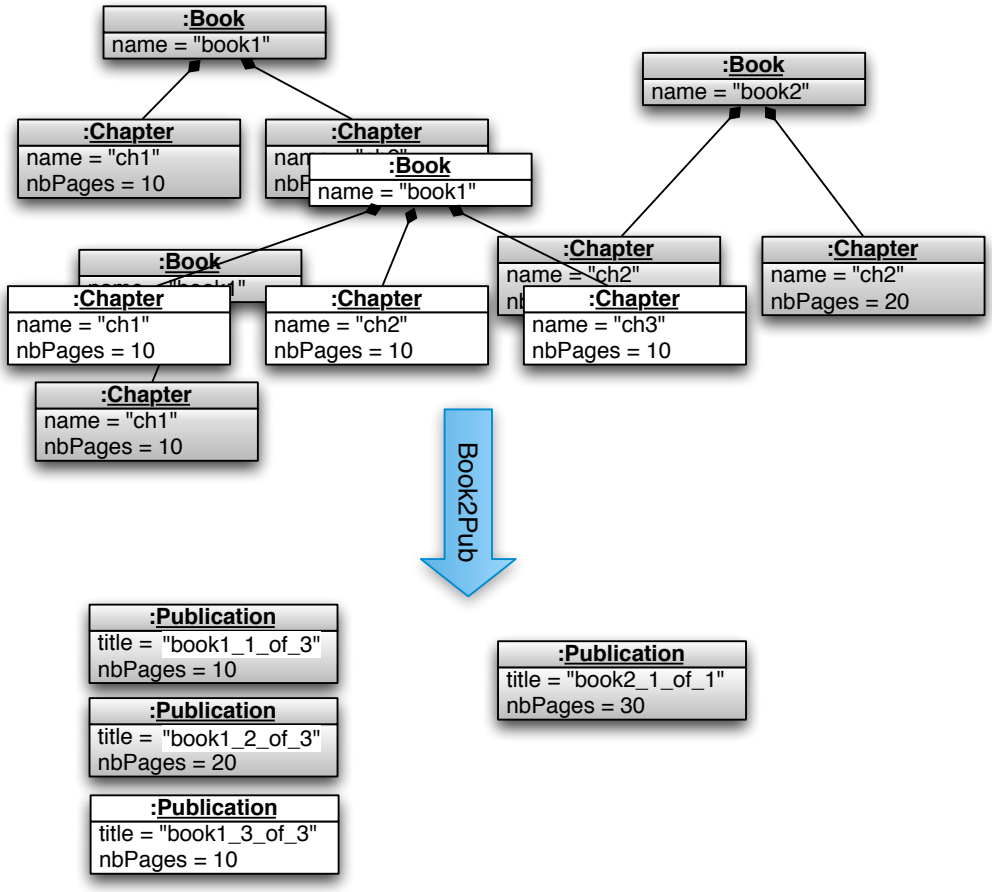


Figure 5.11: Applying the Specialized Transformation to the Changed Source Model

scenario wherein a new book named `book1` is added to the source model. This change entails the insertion of a new instance of `Publication` to the target model as well as updating the `title` attribute of those instances that correspond with older books named `book1` to `boo1_<i>_of_3`, for there exist three such books now.

5.7 Experiments and Discussion

In this section we present the results of our experiments with the partial evaluation framework. We have used the same transformation, i.e., `BookToLibrary` described in the previous section over a range of models of various sizes. Our experiments were performed on a Win-

dows XP (Service pack 3) based PC featuring Intel Core™2 CPU clocked at 2.1Ghz, 2GB of physical memory, running Eclipse M2M project's QVT Operational mapping implementation on top of Eclipse 3.5.

Our first set of experiments involved applying the transformation on models progressively growing in number of elements, and thus in size. More specifically, we used an instance of the `Book` Figure 5.12 compares the performance of the original transformation, and its specialized version. The As it is evident from the graphs, the difference is negligible for small input models. This has to do with I/O being the dominant factor during the transformation of these models, which is comparable for both transformations. However, as we apply partial evaluation on larger models where the processing is the most time-consuming part and the I/O effect is amortized (i.e., models with more than 100 elements), the performance advantage of partial evaluation becomes conspicuous. The details of this experiment is reported in Table 5.1. The input models were generated by a QVT transformation. The reported elements are the ones for which the `BookToLibrary` transformation was partially evaluated. In this set of experiments, we consider the change to be the addition of just one chapter to the first book of the first library. The original transformation requires to transform all other non-affected elements, whereas the partially evaluated one exploits its static cache to expedite the reconciliation of the source and target models. Although at first this minimal size of change for empirical analysis can be called into question, it is in fact representative of a common practical scenario. Time and again, developers have to manipulate bits and pieces of colossal models, and no matter how small the change, the full re-transformation of these models are, more often than not, the only possibilities in many existing modeling tools. Partial evaluation provides a viable solution for these scenarios.

N_T	N_L	N_B	N_C	T_{B2L}	T_{mixB2L}
3	1	1	1	0.12ms	0.11ms
12	1	1	10	0.12ms	0.11ms
111	1	10	10	0.27ms	0.25ms
1111	1	100	10	135.6ms	1.75ms
11011	1	1000	10	10602ms	47ms

Table 5.1: The results of the first set of experiments. N_T: total elements, N_L: no. of `Lib`, N_B: no. of `Book` per `Lib`, N_C: no. of `Chapter` per `Book`, T_{B2L}: Exec. time of `BookToLibrary`, and T_{mixB2L}: Exec. time of `mixBookToLibrary`.

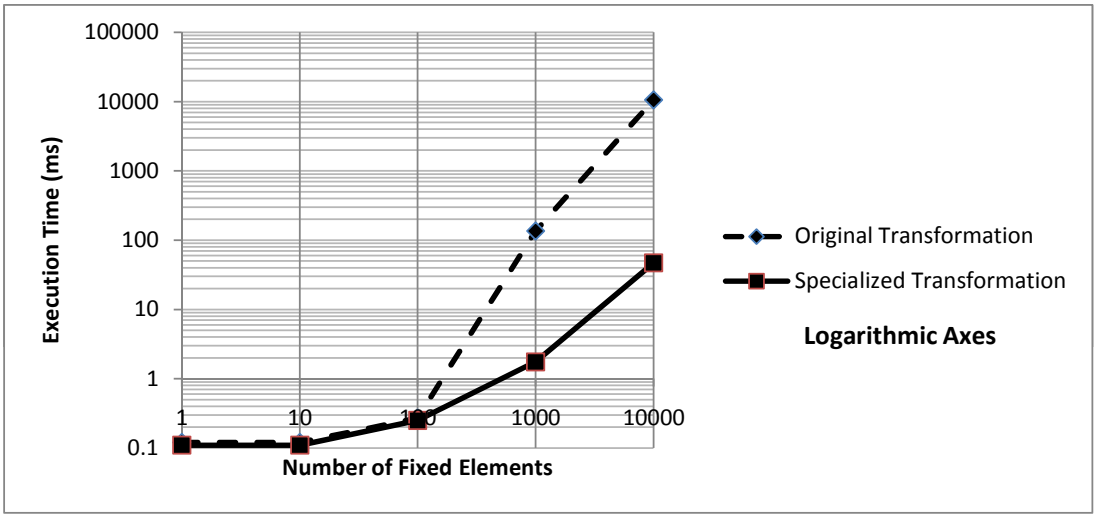


Figure 5.12: Execution time of the original and specialized transformation for growing input sizes

In our second set of experiments, we applied a fixed input model to both transformations, each time instructing the partial evaluator to use a fraction of input for specializing the transformation. This is in effect the same as having the rest of the elements added on the second execution of the transformation. What this experiment assess is whether specializing more expressions inside the program leads to better performance of the specialized program. In particular, we focused in this experiments on the summation of `nbPages` in the program which was reduced by the specializer. The multiple-valued variables were completely cached for each invocation of the transformation. The results of these experiments are projected in Figure 5.13. We triggered the transformation with a model comprised of 10 libraries, 100 books and 10 chapters, the transformation of which took 10703ms by the original `BookToLibrary` transformation. We then varied the percentage of input elements used as FIXED and calculated the time of execution of the transformation specialized for those model elements. As expected, the more input elements were being involved in the partial evaluation, the fewer dynamic computations was need to be performed during re-execution, and thus, the transformation took less time. We started from treating all VAR elements (i.e., instances of Chapter) as dynamic (they can be considered as new elements added after the initial transformation), and reduced this by 10% in each step, which re-

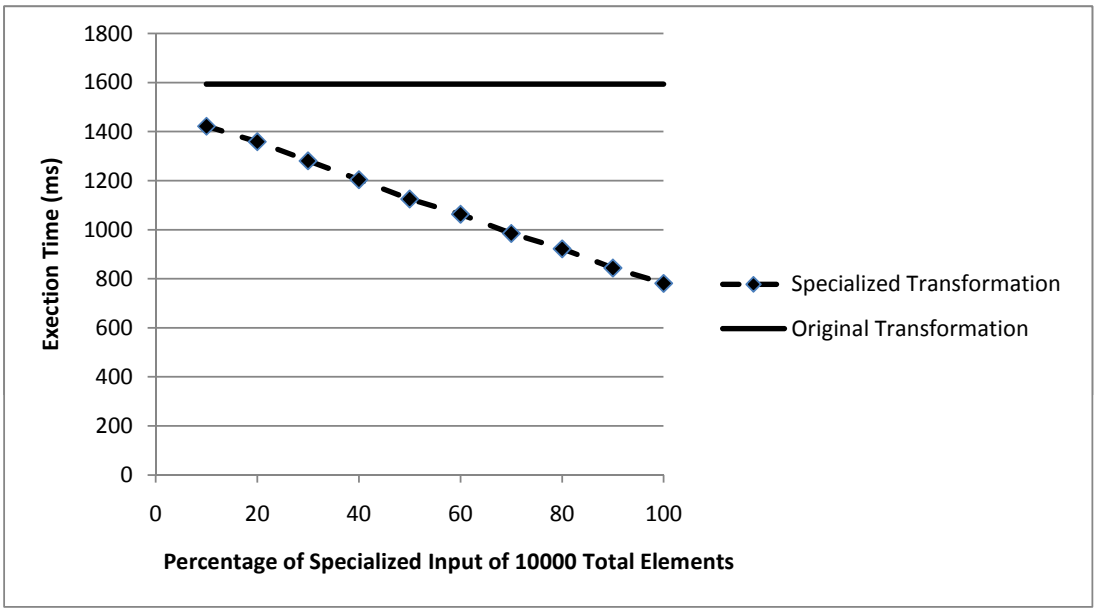


Figure 5.13: Execution time of the original and specialized transformation based on the utilization of input elements for partial evaluation

sulted in more reduction of transformation time. The full partial evaluation performs more than twice as fast as having no partial evaluation.

Chapter 6

Conclusion

This thesis presented an inquiry into various aspects of change management in the context of model driven software engineering. To this end, we laid the foundation for a formalism to denote models and change operations. An algebraic methodology for computing the impact of change operations on models was introduced. We tackled the problem of factorizing the differences of two models into a number of atomic change operators, devising two different algorithms based on two different sets of atomic change operators. We used this foundation to provide a precise characterization for various notions that we discuss in the later sections of the dissertation. More specifically, the classes of monotonic, homomorphic and uniform transformations were introduced and their properties were studied. These properties, i.e., uniformity and locality, allow for generic change translation between source and target sides of a transformation, which lays the foundation for the black-box synchronization approach. We also provided definitions for incrementality and bi-directionality in the context of model transformations.

After introducing the foundation material, the thesis focused on the important problem of model synchronization, for which it proposed a two-fold solution. First, a novel methodology to build synchronization around existing uni-directional and non-incremental transformations was introduced. This technique differs from the previous undertakings primarily in the fact that it uses the original artifact generators as black boxes. As such, other than some generic assumptions about the type of the transformations, it needs no detailed knowledge of the consistency rules. In contrast to the approaches based on incremental transformation engines, our proposed black-box model synchronization framework neither

needs denotation of transformations in a new language, nor it requires the transformations to be re-executed after they are used for the initial creation of target models. The framework, even when used in conjunction with unidirectional artifact generators, is capable of propagating updates in the opposite direction of that of the used artifact generator. The synchronization scheme results in models that comply with the original transformation. The proposed approach is based on a process we refer to as *Conceptualization*. This process extracts the mutual information of two or more inter-related artifacts and stores them in a central concept pool. *Shadow Models* are used as input to the transformations for providing effective traceability between concepts and model elements in interlinked models. We utilize the technique to provide instant and incremental propagation of Update changes between models. To support incremental synchronization of Insertion changes, we propose the notion of μ-templates, some localized place holders in shadow models. Treating transformations as black-boxes has the advantage of eliminating the cost associated with reverse engineering of consistency rules between software artifacts. However, the proposed solution, even though covering a wide spectrum of practical model transformations, is limited to a certain class of model transformations.

To extend the gamut of supported transformations, we introduced a complementary, white-box approach to the problem of model synchronization. In this methodology, we analyze the source code of existing model transformations and transform them to semantically equivalent programs that operate in an incremental fashion. In particular, by leveraging a technique called partial evaluation, we avoid crude re-computation of model transformation rules. We presented QvtMix, a partial evaluator for an essential subset of the QVT Operational Mappings language. Through several examples, we demonstrated the use of QvtMix in different specialization scenarios, during which redundant computations were omitted. Our experiments exhibited significant reduction in re-transformation time as the percentage of the input model elements utilized for partial evaluation increases.

6.1 Future Work

6.1.1 Information Content of Models and Model Transformations

An interesting area for future work is to investigate the notion of Kolmogorov Complexity as a representation for the information content of models and that of transformations.

Informally, the Kolmogorov Complexity of a message is defined as the minimum length of a program that can produce the message as its output. It is shown that the choice of language affects this complexity up to a constant order. In MDE, the notion of representing transformations, which generate models, as model instances themselves has been ubiquitously championed (and in fact is a basis for the white-box methodology presented in this dissertation, too). The Kolmogorov Complexity metric can potentially be a useful metric to categorize various kinds of model transformations based on how much of the information in their source models they propagate to their target side, and to what extent they lose information. Another intriguing investigation we propose is to use this to formulate the relationship of higher-order transformations and the *meta-meta* hierarchies popular in modeling notations such as MOF.

6.1.2 Improved Conceptualization Schemes

The conceptualization scheme proposed here can be improved using information retrieval methods such as Latent Semantics Indexing and Formal Concept Analysis. The idea is to use these methodologies based on existing transformation scenarios to gain insight into the semantic bridges established by the transformation across its domain and co-domain metamodels, and use this knowledge to refine the concepts associated to values in the domain model.

6.1.3 Automatic Generation of Abstractors/DeAbstractors

Model based synchronization techniques are applicable to all types of software artifacts, insofar as they can be represented in a canonical modeling format. The conversion step to create the representation of an artifact, denoted in an arbitrary format, to a canonical representation such as MOF is conducted by so called Abstractors. DeAbstractors do the reverse conversion. A system that can generate these programs automatically (or semi-automatically) from a formal grammar can have immense utility in facilitating the use of model transformations for establishing consistency among myriad of existing formats for software artifacts.

6.1.4 Embedded Rule Language for Composite Concepts

The concept-pool, presented as part of the black-box framework in this dissertation, allows for mappings between concept identifiers and simple values. It, however, can be extended to accommodate expressions that can reference and combine the values of other concepts to increase the expressive power of the framework and provide support for a wider range of model transformations. These embedded rules form a mini-language whose expressions are evaluated by the *deShadow* algorithm. Adding the ability to reference other concepts can, however, create termination issues. We need to investigate methodologies to ensure that the framework terminates, e.g., to provide guarnatees that concept dependencies are cycle-free.

6.1.5 Fine-Grained Version Management Using Concepts

The ideal of a centralized concept-pool and shadow models can have other applications besides model synchronization. One interesting venue is to use this framework to build a configuration management system for maintaining versions of models. A potentially useful advantage of this approach over the existing file-based solutions such as CVS or SVN would be the availability of independent, fine-grained versions down to the level of concepts. This allows us to exploit concepts in order to pick and choose different versions of each element or attribute to form a view of the history of the evolution of a model in the workspace. We envisage that such a feature should have several practical uses.

6.1.6 Enhancements to QvtMix

QvtMix incorporates many important specialization algorithms and in many cases goes beyond what conventional partial evaluators for general purpose programming languages support. Nevertheless, QVT-OM is a language with extensive grammar and several features, which makes the task of extending QvtMix a continuous endeavor. Furthermore, several improvements to the existing specialization scheme are envisioned. Incorporating data-flow analysis algorithms and inter-procedural optimizations common for optimizing compilers can yield more efficient residual programs.

6.1.7 Self Applicability of QvtMix

QvtMix is written *for* and *in* the QVT Operational Mappings language. This is by design: so as to allow for self-applicability. Self-application of partial evaluators, the so called Futamura projections [29], have several interesting theoretical and practical properties. The idea is that partial evaluators can be applied on themselves to generate compilers and compiler generators. This requires, at the very least, that the partial-evaluator be implemented in the same language that it processes. Concocting self-applicable partial evaluators, nonetheless, is known to be a non-trivial task and requires multitude of tweaking and tinkering to make sure certain kinds of symmetries in the code of the partial evaluator are observed.

Appendix A

Haskell Implementation of Change and Sync

Haskell Code

```haskell
module Ecore where

data EObject = EPkg EPackage | ECl EClassifier
data EPackage = EPackage {
                   pkgName      :: EString
                 , nsURI        :: EString
                 , nsPrefix     :: EString
                } deriving Show

data EClassifier =  EClass {
                       clName               :: EString
                     , eStructuralFeatures  :: [EStructuralFeature]
                     , eOperations          :: [EOperation]
                     , eSuperType           :: EString
                     , instanceTypeName     :: EString
                     , abstract             :: EBool
                     , interface            :: EBool
                    }
                  | EDataType {
                       clName               :: EString
                     , clTypeParameters     :: [ETypeParameter]
                    }
                  | EENum {
                       clName               :: EString
                     , eLiterals            :: [EENumLiteral]
                    } deriving Show

data EENumLiteral = EENumLiteral {
                       elitName              :: EString
                     , value                 :: EInt
                    } deriving Show
```

```
data EStructuralFeature =  EAttribute {
                             attName         :: EString
                           , attType         :: EString
                           , defaultValue    :: EString
                           , esLowerBound    :: EInt
                           , esUpperBound    :: EInt
                           , changeable      :: EBool
                           , esOrdered       :: EBool
                           , volatile        :: EBool
                           , unsettable      :: EBool
                           , derived         :: EBool
                           , iD              :: EBool
                           , esUnique        :: EBool
                           , transient       :: EBool
                          }
                         | EReference {
                             refName         :: EString
                           , refType         :: EString
                           , defaultValue    :: EString
                           , containment     :: EBool
                           , derived         :: EBool
                           , eOpposite       :: EString
                           , esLowerBound    :: EInt
                           , esUpperBound    :: EInt
                           , changeable      :: EBool
                           , esOrdered       :: EBool
                           , resolveProxies  :: EBool
                           , transient       :: EBool
                           , unsettable      :: EBool
                           , volatile        :: EBool
                           , esUnique        :: EBool
                          } deriving Show

data EOperation = EOperation {
                    opName          :: EString
                  , opType          :: EString
                  , opLowerBound    :: EInt
                  , opUpperBound    :: EInt
                  , opOrdered       :: EBool
                  , opUnique         :: EBool
                  , eGenericType    :: EString
                  , opTypeParameters :: [ETypeParameter]
                  , eParameters     :: [EParameter]
                 } deriving Show

data EParameter = EParameter {
                    pName           :: EString
                  , pType           :: EString
                  , pOrdered        :: EBool
                  , pUnique         :: EBool
                  , pLowerBound     :: EInt
                  , pUpperBound     :: EInt
                 } deriving Show

newtype ETypeParameter = ETypeParameter EString deriving Show

type EString = String
type EBool = EBoolean
type EBoolean = Bool
type EInt = Int
```

```haskell
class ENamedElement a where
    name  :: a -> EString

class ETypedElement a where
    eType :: a -> EString

class EMultiplicityElement a where
    ordered    :: a -> EBool
    unique     :: a -> EBool
    lowerBound :: a -> EInt
    upperBound :: a -> EInt

class HasETypeParameters a where
    eTypeParameters :: a -> [ETypeParameter]


instance ENamedElement EPackage where
    name = pkgName

instance ENamedElement EClassifier where
    name = clName
instance HasETypeParameters EClassifier where
    eTypeParameters = clTypeParameters

instance ENamedElement EENumLiteral where
    name = elitName

instance ENamedElement EStructuralFeature where
    name (EAttribute {attName = an}) = an
    name (EReference  {refName = rn}) = rn
instance ETypedElement EStructuralFeature where
    eType (EAttribute {attType = at}) = at
    eType (EReference {refType = rt}) = rt
instance EMultiplicityElement EStructuralFeature where
    ordered    = esOrdered
    unique     = esUnique
    lowerBound = esLowerBound
    upperBound = esUpperBound

instance ENamedElement EOperation where
    name  = opName
instance ETypedElement EOperation where
    eType = opType
instance EMultiplicityElement EOperation where
    ordered    = opOrdered
    unique     = opUnique
    lowerBound = opLowerBound
    upperBound = opUpperBound
instance HasETypeParameters EOperation where
    eTypeParameters = opTypeParameters

instance ENamedElement EParameter where
    name  = pName
instance ETypedElement EParameter where
    eType = pType
instance EMultiplicityElement EParameter where
    ordered    = pOrdered
    unique     = pUnique
    lowerBound = pLowerBound
    upperBound = pUpperBound
```

```haskell
isEAttribute :: EStructuralFeature -> Bool
isEAttribute EAttribute {} = True
isEAttribute _ = False

isEContainer :: EStructuralFeature -> Bool
isEContainer EReference {containment = True} = True
isEContainer _ = False

isEReference :: EStructuralFeature -> Bool
isEReference EReference {containment = False} = True
isEReference _ = False
-------------------------------------------------------------------------------
--
-- EcoreParser
--
-------------------------------------------------------------------------------
module EcoreParser where

import Data.Maybe(fromMaybe, fromJust)
import Data.List(isPrefixOf, find)
import Data.Char(toLower)
import Text.XML.Light
import Debug.Trace
import Ecore
import Util

getEPackage :: Element -> Maybe EPackage
getEPackage e = case e of
                 (Element {elName = QName {qName = "EPackage"}}) ->
                    Just EPackage{pkgName = getAttr "name" e
                                  , nsURI   = getAttr "nsURI" e
                                  , nsPrefix = getAttr "nsPrefix" e}
                 otherwise -> Nothing

getEClass :: Maybe String -> Element -> Maybe EClassifier
getEClass xsi e =
        if (elName e == (QName "eClassifiers" Nothing Nothing)) &&
            (getXSIType xsi e == Just "ecore:EClass")
        then Just EClass{ clName =  getAttr "name" e
                   , eStructuralFeatures = getEStructuralFeatures xsi e
                   , eOperations = getEOperations xsi e
                   , eSuperType  = cutPrefix '/' $ getAttr "eSuperTypes" e
                   , instanceTypeName = getAttr "instanceTypeName" e
                   , abstract = case getAttr "abstract" e of
                                  {"true" -> True; _ -> False}
                   , interface = case getAttr "interface" e of
                                  {"true" -> True; _ -> False}
                  }
        else Nothing

getNameSpaces :: Element -> [(String, String)]
getNameSpaces  =
    (map (\a->((qName . attrKey) a, attrVal a))) .
    (filter isNameSpace) .
     elAttribs
    where isNameSpace (Attr (QName _ _ (Just "xmlns")) _ ) = True
          isNameSpace _ = False

xsiURI :: Element -> Maybe String
xsiURI e = findXSI (getNameSpaces e)>>= Just . snd
    where findXSI = find (\(k,v) -> k == "xsi")
```

```
217 getXSIType :: Maybe String -> Element -> Maybe String
218 getXSIType xsi = findAttr (QName "type" xsi (Just "xsi"))

220 getAttr at =
221     (fromMaybe "" ) .
222       (findAttr (unqual at))

224 getEStructuralFeatures :: Maybe String -> Element -> [EStructuralFeature]
225 getEStructuralFeatures xsi =
226     let featElems = findElements (unqual "eStructuralFeatures")  in
227         map (getEStructuralFeature xsi) . featElems

229 getEStructuralFeature :: Maybe String -> Element -> EStructuralFeature
230 getEStructuralFeature xsi e =
231     case getXSIType xsi e of
232         Just "ecore:EAttribute" -> makeAttribute e
233         Just "ecore:EReference" -> makeReference e
234     where
235         makeAttribute e =
236             EAttribute
237             {
238                attName = getAttr "name" e
239              , attType = cutPrefix '/' $ getAttr "eType" e
240              , defaultValue = getAttr "defaultValue" e
241              , esLowerBound = toInt (getAttr "lowerBound" e)
242              , esUpperBound = toInt (getAttr "upperBound" e)
243              , changeable = toBool (getAttr "changeable" e)
244              , esOrdered = toBool (getAttr "ordered" e)
245              , volatile = toBool (getAttr "volatile" e)
246              , unsettable = toBool (getAttr "unsettable" e)
247              , derived = toBool (getAttr "derived" e)
248              , iD = toBool (getAttr "iD" e)
249              , esUnique = toBool (getAttr "unique" e)
250              , transient = toBool (getAttr "transient" e)
251             }
252         makeReference e =
253             EReference
254             {
255                refName = getAttr "name" e
256              , refType = cutPrefix '/' $ getAttr "eType" e
257              , defaultValue = getAttr "defaultValue" e
258              , containment = toBool (getAttr "containment" e)
259              , derived = toBool (getAttr "derived" e)
260              , eOpposite = getAttr "eOpposite" e
261              , esLowerBound = toInt (getAttr "lowerBound" e)
262              , esUpperBound = toInt (getAttr "upperBound" e)
263              , changeable = toBool (getAttr "changeable" e)
264              , esOrdered = toBool (getAttr "ordered" e)
265              , resolveProxies = toBool (getAttr "resolveProxies" e)
266              , transient = toBool (getAttr "transient" e)
267              , unsettable = toBool (getAttr "unsettable" e)
268              , volatile = toBool (getAttr "volatile" e)
269              , esUnique = toBool (getAttr "unique" e)
270             }

272 getEOperations :: Maybe String -> Element -> [EOperation]
273 getEOperations xsi =
274     let opElems = findElements (unqual "eOperations")  in
275         map (getEOperation xsi) . opElems
```

```haskell
getEOperation :: Maybe String -> Element -> EOperation
getEOperation xsi e =  EOperation {
                          opName = getAttr "name" e
                        , opType = cutPrefix '/' $ getAttr "eType" e
                        , opLowerBound = toInt (getAttr "lowerBound" e)
                        , opUpperBound = toInt (getAttr "upperBound" e)
                        , opOrdered = toBool (getAttr "ordered" e)
                        , opUnique = toBool (getAttr "ordered" e)
                        , eGenericType = getAttr "eGenericType" e
                        , opTypeParameters = getETypeParameters e
                        , eParameters = getEParameters e
                       }

getETypeParameters :: Element -> [ETypeParameter]
getETypeParameters  = let tparElem = findElements (unqual "eTypeParameters") in
                          map getETypeParameter . tparElem


getETypeParameter :: Element -> ETypeParameter
getETypeParameter e = ETypeParameter $ getAttr "name" e

getEParameters :: Element -> [EParameter]
getEParameters = let parElem = findElements (unqual "eParameters") in
                     map getEParameter . parElem

getEParameter :: Element -> EParameter
getEParameter e = EParameter {
                      pName = getAttr "name" e
                    , pType = cutPrefix '/' $ getAttr "eType" e
                    , pOrdered = toBool (getAttr "ordered" e)
                    , pUnique = toBool (getAttr  "unique" e)
                    , pLowerBound = toInt (getAttr "lowerBound" e)
                    , pUpperBound = toInt (getAttr "upperBound" e)
                   }

-------------------------------------------------------------------------------
--
-- Model
--
-------------------------------------------------------------------------------
module Model where

import Ecore
import EcoreParser
import Text.XML.Light
import Data.Maybe
import Data.List
import Util
import Debug.Trace
import Control.Monad.Reader
import System.IO

test = do
        input <- readFile "JavaSrvImpl.ecore.xml"
        xmi <- readFile "JavaClass.xmi"
        let elems = onlyElems (parseXML input)
            pkgEl = head $ tail elems
            epkg  = fromJust $ getEPackage pkgEl
            children = elChildren pkgEl
            ecla = map fromJust $
                    filter (\m-> case m of
```

```
                                {(Just _) -> True; Nothing -> False}) $
                    map (getEClass (xsiURI pkgEl)) children
            metaModel = getMetaModel "JavaSrvImpl" ecla
            topElem = head $ tail $ onlyElems $ parseXML xmi

        print topElem
        putStrLn "--------------------"
        print metaModel
        putStrLn "--------------------"
        let mod = xmiMMToModel metaModel (getElementType topElem) topElem
            addr = (Root mod) </> 0 <.> 0

        putStrLn "--------------------"
        print mod
        print addr
        print $ getModel addr
        return mod

------------------------------------------------------------
-- MetaModel
------------------------------------------------------------

data MetaModel = MetaModelElement {
                    mmeName :: String
                  , mmeContainers :: [(Container, Type)]
                  , mmeAttributes :: [(Attribute, Type)]
                  , mmeReferences :: [(Reference, Type)]
                  , mmeSuper :: String
                 } | MetaModel {
                       mmName::String
                     , mmElems::[MetaModel]
                     } deriving (Show, Read)
------------------------------------------------------------
-- Model
------------------------------------------------------------

data Model = Model  {
                mName :: String
              , mContainers :: [[Model]]
              , mAttributes :: [Attribute]
              , mReferences :: [Reference]
              , mType :: Type
            } deriving (Show, Read, Eq)

------------------------------------------------------------
-- Addr
------------------------------------------------------------

data Addr = Root Model | Cont Addr Int| El Addr Int|At Addr Int

model m = Root $ Model {mName = m}

------------------------------------------------------------
-- Container Selector
------------------------------------------------------------

(</>) :: Addr -> Int -> Addr
a </> i = Cont a i

------------------------------------------------------------
-- Element Selector
```

```
399 ------------------------------------------------------------

401 (<.>) :: Addr -> Int -> Addr
402 a <.> i = El a i

404 ------------------------------------------------------------
405 -- Attribute Selector
406 ------------------------------------------------------------

408 (<@>) :: Addr -> Int -> Addr
409 a <@> i = At a i

411 getAt :: Addr -> Maybe Attribute
412 getAt (At addr i) = do
413           m <- getModel addr
414           let attribs = mAttributes m
415           if i < length attribs then Just $ attribs !! i else Nothing
416 getAt _ = Nothing

418 getCont :: Addr -> Maybe [Model]
419 getCont (Cont addr i) = do
420          m <- getModel addr
421          let conts = mContainers m
422          if  i < length conts then Just $ conts !! i else Nothing
423 getCont _ = Nothing

425 getEl :: Addr -> Maybe Model
426 getEl e@(El _ _) = getModel e
427 getEl _ = Nothing

429 getModel :: Addr -> Maybe Model
430 getModel (Root m) = Just m
431 getModel (Cont a i) = getModel a
432 getModel (At a i) = getModel a
433 getModel (El (Cont a c) e)  = getModel a >>= getEl c e
434     where
435         getEl :: Int -> Int -> Model -> Maybe Model
436         getEl c e m = let conts = mContainers m
437                           elems = conts !! c
438                       in
439                        if length conts > c && length elems > e then
440                             Just $ elems !! e
441                        else
442                             Nothing

444 instance Show Addr where
445   show (Root m) = "/" ++ mName m
446   show (Cont addr c) = (show addr) ++  '/':(show c)
447   show (El addr e) = (show addr) ++  '.':(show e)
448   show (At addr a) = (show addr) ++ '@':(show a)

450 type Container = String
451 type Attribute = String
452 type Reference = String
453 type Type = String

455 emptyModel :: Model
456 emptyModel = Model "" [] [] [] "_Nil_"

458 getMetaModel :: String -> [EClassifier] -> MetaModel
459 getMetaModel name ecla =
```

```
460     MetaModel name $ map fromEcore ecla

462 fromEcore :: EClassifier -> MetaModel
463 fromEcore (EClass name  features _ super _ _ _) =
464    MetaModelElement name c a r super
465   where
466     c = map (\f-> (refName f, refType f)) $
467         filter isEContainer features
468     a = map (\f-> (attName f, attType f)) $
469         filter isEAttribute features
470     r = map (\f-> (refName f, refType f)) $
471         filter isEReference features

473 xmiToModel ::  Element -> Model
474 xmiToModel e = Model {
475                         mName = getElementName e
476                       , mContainers = getContainers e
477                       , mAttributes = getAttributes e
478                       , mReferences = getReferences e
479                       , mType = getElementType e
480                      }
481 xmiMMToModel ::  MetaModel -> String -> Element -> Model
482 xmiMMToModel mm ty e =
483     let mme = find (\m -> mmeName m == ty) (mmElems mm)
484      in case mme of
485       Just mme' ->
486          Model {
487             mName = getElementName e
488           , mContainers = getMMContainers mm mme' e
489           , mAttributes = getMMAttributes mme' e
490           , mReferences = getMMReferences mme' e
491           , mType = ty
492          }
493       otherwise -> emptyModel

495 getElementName :: Element -> String
496 getElementName  = getAttr "name"

498 getAttributes :: Element -> [Attribute]
499 getAttributes = (map attrVal) . (filter sansPrefix) . elAttribs
500                 where
501                    sansPrefix (Attr (QName _ _ Nothing) _) = True
502                    sansPrefix _ = False

504 getMMAttributes :: MetaModel -> Element -> [Attribute]
505 getMMAttributes mme e = [attrVal a | (k,t) <- mmeAttributes mme,
506                                      a <- elAttribs e,
507                                      qName (attrKey a) == k]

509 getElementType :: Element -> Type
510 getElementType = qName . elName

512 getReferences :: Element -> [Reference]
513 getReferences _ = []

515 getMMReferences :: MetaModel -> Element -> [Reference]
516 getMMReferences mme e = [attrVal a | (k,t) <- mmeReferences mme,
517                                     a <- elAttribs e,
518                                     qName (attrKey a) == k]

520 getContainers :: Element -> [[Model]]
```

```haskell
getContainers =  (map (map xmiToModel)) .
                  (groupBy (\e1 e2-> elName e1 == elName e2)) .
                    elChildren

getMMContainers ::  MetaModel -> MetaModel -> Element -> [[Model]]
getMMContainers mm mme e = [[xmiMMToModel mm t child] | (cont,t) <- mmeContainers mme, child <- elChildren e, cont ==
      qName(elName child)]

--mapModel f (Nil a t) = Nil (map f a) t
--mapModel f (Model ms a t) = Model (map (map (mapModel f)) ms) (map f a) t

storeLazyModel :: Model -> String -> IO ()
storeLazyModel model path = writeFile path $ show model

loadLazyModel :: String -> IO Model
loadLazyModel path = readFile path >>= return.read

storeModel :: Model -> String -> IO ()
storeModel model path =
    do
        handle <- openFile path WriteMode
        hPutStr handle $ show model
        hClose handle

loadModel :: String -> IO Model
loadModel path =
    do
        handle <- openFile path ReadMode
        content <- loadContent handle ""
        hClose handle
        return $ read content
    where loadContent h str =
                do
                    eof <- hIsEOF h
                    if eof
                        then return str
                        else do line <- hGetLine h
                                let newS = str++line
                                n

-------------------------------------------------------------------------
--
-- Change
--
-------------------------------------------------------------------------

import Model
import Concept

m = (Model "M" [[(Model "C1" [] ["c1attr1","c1attr2"] [] "C")]] ["m1a1"] [] "MC")

-------------------------------------------------------------
-- Update
-------------------------------------------------------------

upd :: Attribute -> Addr -> Maybe Model
upd v (At addr i) =  getModel addr >>= return.updElem i v
upd _ _ = Nothing


updElem :: Int -> Attribute -> Model -> Model
```

```
updElem  i v m =
    if  i >= 0 && i < length attrs && v /= oldVal then
       m{mAttributes = newAttrs}
    else
       m
    where attrs = mAttributes m
          oldVal = attrs !! i
          newAttrs = let (xs, v':ys) = splitAt i attrs in
                        xs++v:ys

------------------------------------------------------------
-- Insert
------------------------------------------------------------

ins :: Type -> Addr -> Maybe Model
ins t (Cont addr i) = getModel addr >>= return.insElem t i
ins _ _ = Nothing


insElem ::  Type -> Int -> Model -> Model
insElem t i m =
    if i >= 0 && i < length containers then
        m {mContainers = newContainers}
    else if i == 0 && length containers == 0 then
           insElem t 0 m {mContainers = [[]]}
         else
           m
    where containers = mContainers m
          newContainers = let (cs, ci:cs') = splitAt i containers in
                             cs++(ci++[emptyModel{mType = t}]):cs'

------------------------------------------------------------
-- Delete
------------------------------------------------------------

del :: Addr -> Maybe Model
del (Cont addr i) = getModel addr >>= return.delElem i
del _ = Nothing

delElem :: Int -> Model -> Model
delElem i m =
    if i >= 0 && i < length containers && not (null $ containers !! i)  then
        m{mContainers = newContainers}
    else
        m
    where containers = mContainers m
          newContainers = let (cs, ci:cs') = splitAt i containers  in
                             cs++(init ci):cs'


------------------------------------------------------------------------------
--
-- Concept
--
------------------------------------------------------------------------------
module Concept where

import Control.Monad.State
import System.Time
import Data.Map
```

```
-------------------------------------------------------------
-- Concept
-------------------------------------------------------------

data Concept =  Concept {
                    cID :: ConceptID
                  , cVal :: String
                --  , refs :: [Addr]
                 -- , history :: [(TimeStamp, String)]
                 -- , newMu :: Maybe Concept
                  }-- | Mu { id :: ConceptID } | Nil { id:: ConceptID}
                   deriving (Show)

type ConceptID = String
type TimeStamp = ClockTime

-------------------------------------------------------------
-- ConceptPool
-------------------------------------------------------------

type ConceptPool = Map ConceptID Concept
type ConState = StateT ConceptPool IO

newID :: String
newID = undefined


updateCon :: ConceptID -> String -> ConState ()
updateCon cid v =
    get >>=  put.update (\c -> Just c{cVal = v}) cid

addCon :: Concept -> ConState ()
addCon c = get >>= put.insert (cID c) c

getCon ::  ConceptID  -> ConState (Maybe Concept)
getCon cid = get >>=  return . Data.Map.lookup cid

-------------------------------------------------------------------------------
--
-- Shadow
--
-------------------------------------------------------------------------------
module Shadow where

import Ecore
import EcoreParser
import Model
import Concept
import Data.UUID.V1
import Control.Monad.Trans
import Control.Monad.State
import Data.Maybe
import Debug.Trace
import qualified Data.Map as Map

type ShadowMap = [(Model, Model)]
type ShadowFileMap = [(String, String)]

data ShadowState = ShadowState {  shMemMap ::ShadowMap
                                , shFileMap:: ShadowFileMap} deriving(Show)
```

```haskell
type ShadowStateT = StateT ShadowState ConState

mapModel f (Model n c a r t) =
    Model n (map  (map (mapModel f)) c) (map f a) r t

mapModelM :: (Monad m) => (String -> m String) -> Model -> m Model
mapModelM f (Model n c a r t) =
    do
     fa <- mapM f a
     fc <- mapM (mapM (mapModelM f)) c
     return $ Model n fc fa r t

shadowAttr v = do
    cuuid <- liftIO nextUUID
    let cid = show `fmap` cuuid
    unless (cid == Nothing) $ addCon (Concept (fromJust cid)  v)
    return cid

-------------------------------------------------------------
-- Shadow
-------------------------------------------------------------

shadow m = mapModelM (\a -> shadowAttr a >>= return.fromJust) m

-------------------------------------------------------------
-- DeShadow
-------------------------------------------------------------

deShadow s = mapModelM (\cid -> (getCon cid) >>= return.cVal.fromJust) s

createShadowModel :: String -> ShadowStateT ()
createShadowModel modelPath =
    do
        model <- liftIO $ loadLazyModel modelPath
        shadowModel <- lift $ shadow model
        shMap <- get
        let shadowPath = (takeWhile (/='.') modelPath) ++ "Shadow.mod"
        let newShadowState = shMap {
                shMemMap = (model,shadowModel):(shMemMap shMap )
              , shFileMap = (modelPath,shadowPath):(shFileMap shMap)
              }
        put newShadowState
        liftIO $ storeLazyModel shadowModel shadowPath
        state <- get
        liftIO $ putStrLn (show state)

testShadow = runStateT (runStateT testScript (ShadowState [] [])) Map.empty
    where testScript =
            do
                createShadowModel "m.mod"
                m <-liftIO $ loadLazyModel "mShadow.mod"
                liftIO $ putStrLn $ "Shadow Model:    " ++ (show m)
                dm <- lift $ deShadow m
                liftIO $ putStrLn $ "DeShadow Model:     " ++ (show dm)

-------------------------------------------------------------------------------
--
-- Util
--
-------------------------------------------------------------------------------
module Util where
```

```haskell
import Data.Char

splitWith :: (Char -> Bool) -> String -> [String]
splitWith _ [] = []
splitWith p xs = let (ts, fs) = break p xs
                    in ts:splitWith p (tail' fs)
                 where
                    tail' [] = []
                    tail' (x:xs) = xs

cutPrefix c = last' . (splitWith (\x->x==c))
              where last' [] = []
                    last' xs = last xs

toBool s = case map toLower s of
                "true" -> True
                otherwise -> False
toInt s = case reads s::[(Int, String)] of
                [(n, [])] -> n
                otherwise -> 0
```

Appendix B

Abstract Syntax of QVT Operational Mappings

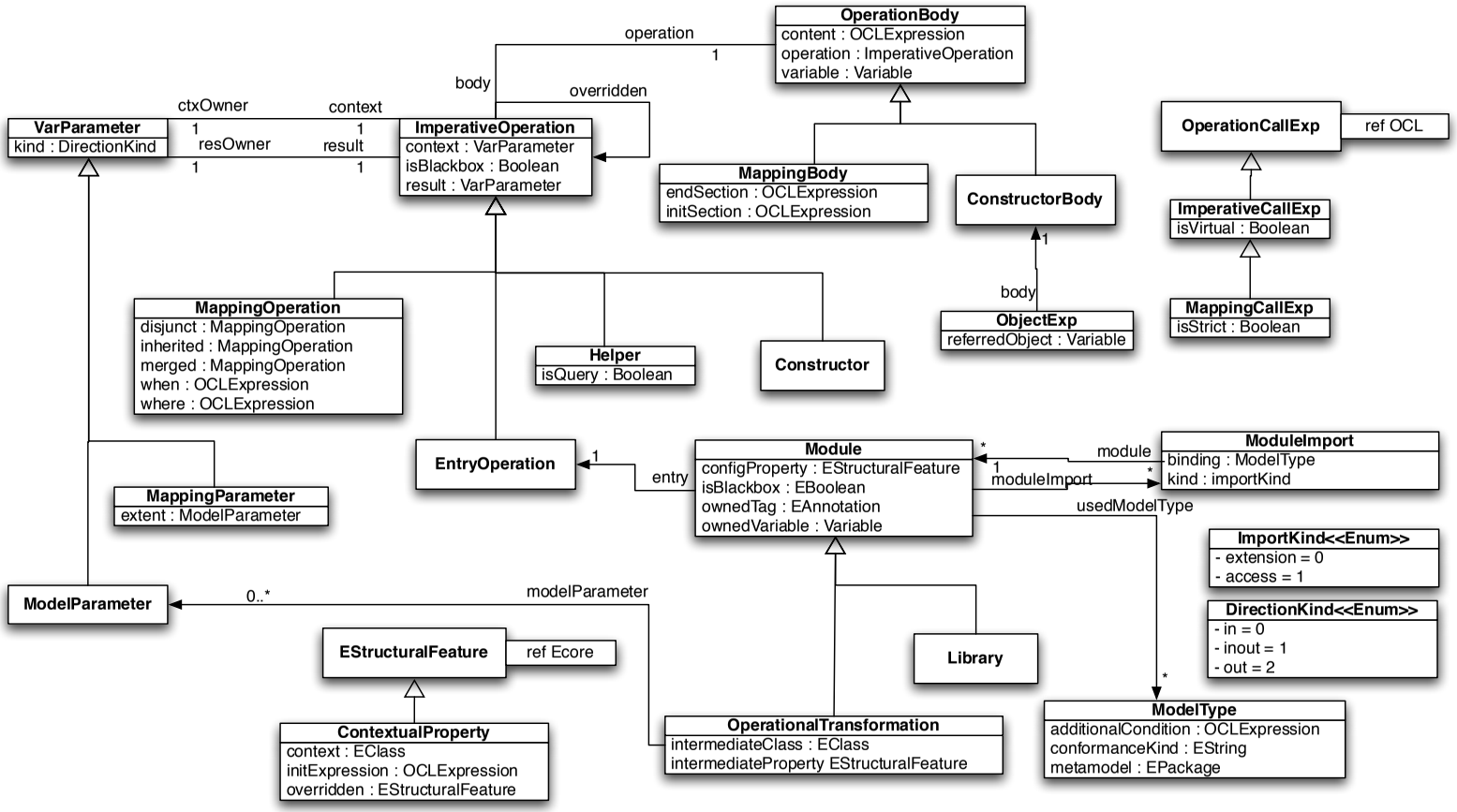


Figure B.1: Simplified Metamodel of QVT Operational Mappings

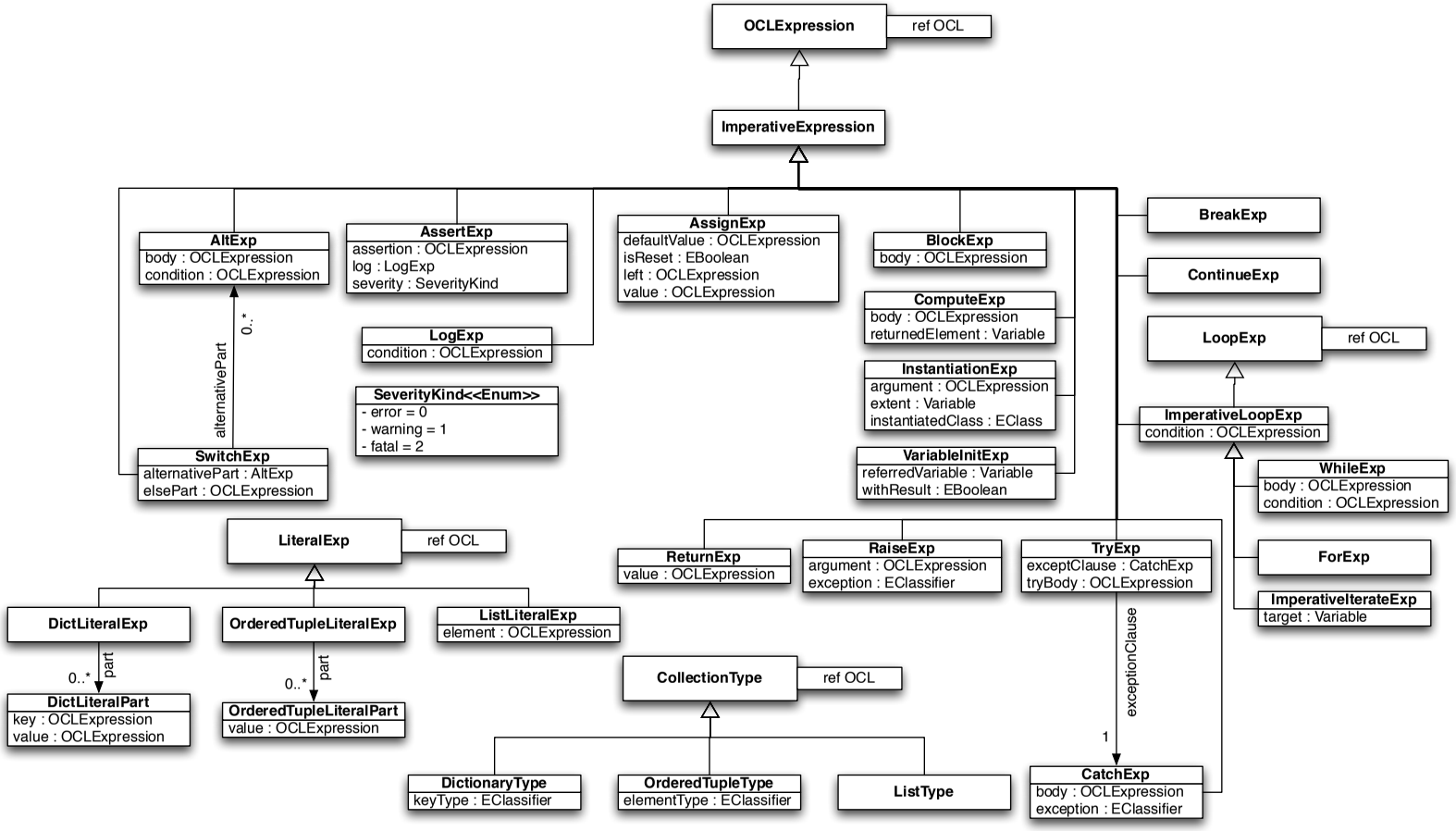


Figure B.2: Simplified Metamodel of Imperative OCL

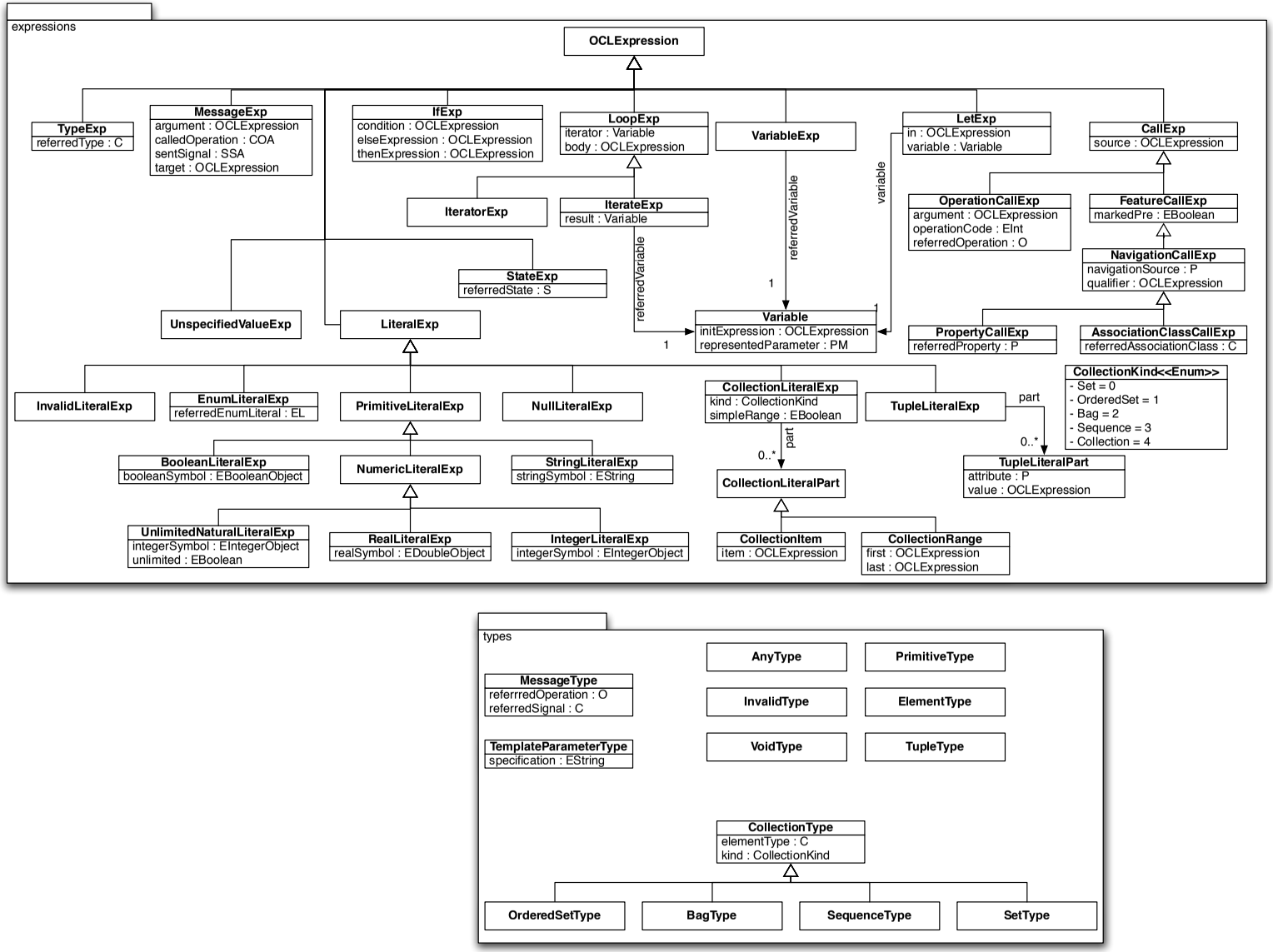


Figure B.3: Simplified Metamodel of OCL

Appendix C

Big-Step Operational Semantics of QVT Operational Mappings

C.1 Big-Step Operational Semantics

In this section we present a formal semantics for Essential QVT-OM. The method chosen for denoting the semantics of the language is Big-Step Operational Semantics. Further information on specifying formal semantics of programming languages can be found in standard graduate level references on programming languages theory [79, 65, 59]. QVT-OM has a number of unique constructs that are not present in other languages. In that respect, a formal semantics for the language can be immensely useful to analyze profoundly the effect of these unfamiliar features. The semantics rules that deal with models and change operations use the formalism presented in Chapter 3.

The first set of rules spell out the meaning of literal values, constant collections, model element attribute, container and references access and basic arithmetic operations.

$$\frac{\sigma(x) = v}{\sigma; x \Downarrow v}\ E-Var \qquad \frac{\langle \sigma, e\rangle \Downarrow \langle \sigma', a\rangle}{\langle \sigma, e.\texttt{-}() \rangle \Downarrow \langle \sigma', -a\rangle}\ E-Neg$$

$$\frac{\langle \sigma, e_0\rangle \Downarrow \langle \sigma_0, v_0\rangle\ \langle \sigma_{i-1}, e_i\rangle \Downarrow \langle \sigma_i, v_i\rangle|_{1\leqslant i\leqslant n}\ \langle \sigma_n, v_0.f(v_1,..,v_n)\rangle \Downarrow \langle \sigma', v'\rangle}{\langle \sigma, e_0.f(e_1,..,e_n)\rangle \Downarrow \langle \sigma', v'\rangle}\ E-Call$$

$$\frac{\sigma; e_1 \Downarrow a \qquad \sigma; e_2 \Downarrow b}{\begin{array}{ll}\sigma; e_1.\texttt{+}(e_2) \Downarrow a+b & \sigma; e_1.\texttt{-}(e_2) \Downarrow a-b \\ \sigma; e_1.\texttt{*}(e_2) \Downarrow a\times b & \sigma; e_1./(e_2) \Downarrow e_1 \div e_2\end{array}}\ E-Op \qquad \frac{}{\sigma; \underline{\text{true}} \Downarrow True}\ E-True$$

$$\frac{}{\sigma; \underline{\text{false}} \Downarrow False}\ E-False \qquad \frac{}{\sigma; \underline{\text{Set}}\{\} \Downarrow \varnothing}\ E-EmptySet$$

$$\frac{\sigma; e_i \Downarrow v_i|_{1\leqslant i\leqslant n}}{\sigma; \underline{\text{Set}}\{e_1,..,e_n\} \Downarrow \cup_{i=1}^{n}\{v_i\}}\ E-Set \qquad \frac{}{\sigma; \underline{\text{OrderedSet}}\{\} \Downarrow \langle\rangle}\ E-EmptyOrderedSet$$

$$\frac{\sigma; e_i \Downarrow v_i|_{1\leqslant i\leqslant n}}{\sigma; \underline{\text{OrderedSet}}\{e_1,..,e_n\} \Downarrow \langle v_1,..,v_n\rangle}\ E-OrderedSet$$

$$\frac{}{\sigma; \underline{\text{Dict}}\{\} \Downarrow \varnothing}\ E-EmptyDict$$

$$\frac{\sigma; k_i \Downarrow \mu_i|_{1\leqslant i\leqslant n} \qquad \sigma; e_i \Downarrow v_i|_{1\leqslant i\leqslant n}}{\sigma; \underline{\text{Dict}}\{k_1 = e_1,..,k_n = e_n\} \Downarrow \lambda x. \begin{cases} v_1 & x = \mu_1 \\ \vdots & \\ v_n & x = \mu_n \\ \bot & otherwise \end{cases}}\ E-Dict$$

$$\frac{\sigma; e \Downarrow (\mathfrak{C}, \langle a_1,..,a_n\rangle, R, T) \qquad \sigma(T) = (Sig_C, Sig_A, Sig_R)\ \exists\ i\ Sig_A(i) = (\texttt{"attr"}, T_a)}{\sigma; e.\texttt{attr} \Downarrow a_i}\ E-Attr$$

$$\frac{\sigma; e \Downarrow (\langle \mathfrak{C}_1,..,\mathfrak{C}_n\rangle, A, R, T) \qquad \sigma(T) = (Sig_C, Sig_A, Sig_R)\ \exists\ i\ Sig_C(i) = (\texttt{"cont"}, T_c)}{\sigma; e.\texttt{cont} \Downarrow \mathfrak{C}_i}\ E-Cont$$

$$\frac{\begin{array}{l}\sigma; e \Downarrow (\mathfrak{C}, A, \langle r_1,..,r_n\rangle, T) \quad \sigma(T) = (Sig_C, Sig_A, Sig_R) \\ \exists\ i\ Sig_R(i) = (\texttt{"ref"}, T_R) \quad \sigma(r_i) = (\mathfrak{C}_r, A_r, R_r, T_r)\end{array}}{\sigma; e.\texttt{ref} \Downarrow (\mathfrak{C}_r, A_r, R_r, T_r)}\ E-Reference$$

$$\frac{\sigma; e \Downarrow \nu \qquad \sigma(x) = \bot}{\langle \sigma, \underline{\mathsf{var}}\ \mathsf{x}\ \mathsf{:=}\ e\rangle \Downarrow \langle \sigma[x \mapsto \nu], \nu\rangle}\ \mathrm{E-Init} \qquad \frac{\sigma; e \Downarrow \nu \qquad \sigma(x) \neq \bot}{\langle \sigma, \mathsf{x}\ \mathsf{:=}\ e\rangle \Downarrow \langle \sigma[x \mapsto \nu], \nu\rangle}\ \mathrm{E-Assign}$$

$$\frac{\sigma; e_{init} \Downarrow \nu}{\langle \sigma, \underline{\mathsf{property}}\ \mathsf{p}\ \mathsf{:}\ \mathsf{T}\ \mathsf{=}\ e_{init}\rangle \Downarrow \langle \sigma[p \mapsto \nu], \_\rangle}\ \mathrm{E-Property}$$

$$\frac{\sigma; e_{cond} \Downarrow \mathrm{True} \qquad \sigma; e_{then} \Downarrow \nu_{then}}{\sigma; \underline{\mathsf{if}}\ e_{cond}\ \underline{\mathsf{then}}\ e_{then}\ \underline{\mathsf{else}}\ e_{else} \Downarrow \nu_{then}}\ \mathrm{E-If-True}$$

$$\frac{\sigma; e_{cond} \Downarrow \mathrm{False} \qquad \sigma; e_{else} \Downarrow \nu_{else}}{\sigma; \underline{\mathsf{if}}\ e_{cond}\ \underline{\mathsf{then}}\ e_{then}\ \underline{\mathsf{else}}\ e_{else} \Downarrow \nu_{else}}\ \mathrm{E-If-False}$$

$$\frac{}{\sigma; \underline{\mathsf{new}}\ \mathsf{T()} \Downarrow (\varnothing, \varnothing, \varnothing, \mathsf{T})}\ \mathrm{E-New}$$

$$\frac{\langle \sigma_{i-1}, e_i\rangle \Downarrow \langle \sigma_i, \nu_i\rangle|_{1 \leqslant i \leqslant n}}{\sigma, \underline{\mathsf{object}}\ \mathsf{T}\ \{\mathsf{a}_1 := e_1; ..; \mathsf{a}_n := e_n;\} \Downarrow (\varnothing, \langle \nu_1, .., \nu_n\rangle, \varnothing, \mathsf{T})}\ \mathrm{E-Object}$$

$$\frac{\sigma; e_{cond} \Downarrow \mathrm{False}}{\langle \sigma, \underline{\mathsf{while}}\ e_{cond}\ \underline{\mathsf{do}}\ e_{body}\rangle \Downarrow \langle \sigma, \_\rangle}\ \mathrm{E-While-False}$$

$$\frac{\begin{array}{l}\sigma; e_{cond} \Downarrow \mathrm{True} \\ \langle \sigma, \wp, e_{body}\rangle \Downarrow \langle \sigma', \wp, \_\rangle \\ \langle \sigma', \wp, \underline{\mathsf{while}}\ e_{cond}\ \underline{\mathsf{do}}\ e_{body}\rangle \Downarrow \langle \sigma'', \_, \_\rangle\end{array}}{\langle \sigma, \wp, \underline{\mathsf{while}}\ e_{cond}\ \underline{\mathsf{do}}\ e_{body}\rangle \Downarrow \langle \sigma'', \_, \_\rangle}\ \mathrm{E-While-True}$$

$$\frac{}{\langle \sigma, \wp, \underline{\mathsf{break}}; e\rangle \Downarrow \langle \sigma, \lightning, \_\rangle}\ \mathrm{E-Break}$$

$$\frac{}{\langle \sigma, \lightning, \underline{\mathsf{while}}\ e_{cond}\ \underline{\mathsf{do}}\ e_{body}\rangle \Downarrow \langle \sigma, \_, \_\rangle}\ \mathrm{E-While-Break}$$

$$\frac{\sigma; e_s \Downarrow \varnothing}{\langle \sigma, e_s\text{->}\underline{\mathsf{forEach}}\mathsf{(x)}\ e_{body}\rangle \Downarrow \langle \sigma, \_\rangle}\ \mathrm{E-For-Empty}$$

$$\frac{\langle \sigma, e_s\rangle \Downarrow \langle \sigma_0, \langle a_1, .., a_n\rangle\rangle \qquad \langle \sigma_{i-1}[x \mapsto a_i], e_{body}\rangle \Downarrow \langle \sigma_i, \_\rangle|_{1 \leqslant i \leqslant n}}{\langle \sigma, e_s\text{->}\underline{\mathsf{forEach}}\mathsf{(x)}\ e_{body}\rangle \Downarrow \langle \sigma_n, \_\rangle}\ \mathrm{E-For}$$

$$\frac{}{\begin{array}{c}\langle \zeta, \underline{\texttt{helper}}\ \mathsf{T_s} :: \texttt{f(}\mathsf{a_1} : \mathsf{T_1}, .., \mathsf{a_n} : \mathsf{T_n}\texttt{)}\ \texttt{:}\ \ \mathsf{T_r}\ e_{body}\rangle \\ \Downarrow \\ \langle \zeta[\texttt{f} \mapsto \zeta(\texttt{f}) \cup \{(\mathsf{T_s}, \mathsf{T_r}, \langle(\mathsf{a_i}, \mathsf{T_i})\rangle, e_{body})\}], \_\rangle\end{array}}\ E-Helper$$

$$\frac{}{\begin{array}{c}\langle \xi, \underline{\texttt{mapping}}\ \mathsf{T_s} :: \texttt{m(}\mathsf{a_1} : \mathsf{T_1}, .., \mathsf{a_n} : \mathsf{T_n}\texttt{)}\ \texttt{:}\ \ \mathsf{T_r}\ e_i\ e_p\ e_e\rangle \\ \Downarrow \\ \langle \xi[\texttt{m} \mapsto \xi(\mathsf{m}) \cup \{(\mathsf{T_s}, \mathsf{T_r}, \langle(\mathsf{a_i}, \mathsf{T_i})\rangle, e_i, e_p, e_e)\}], \_\rangle\end{array}}\ E-Mapping$$

$$\frac{\langle(\sigma_p, \sigma), e\rangle \Downarrow \nu}{\langle(\sigma_p, \sigma), \downarrow, \underline{\texttt{return}}\ e\rangle \Downarrow \langle(\sigma_p, \sigma), \lnot, \nu\rangle}\ E-Return$$

$$\frac{\langle(\sigma_p, \sigma), \downarrow, e\rangle \Downarrow \langle(\sigma'_p, \sigma', \downarrow, \nu\rangle \quad \langle(\sigma'_p, \sigma'), \downarrow, e'\rangle \Downarrow \langle(\sigma''_p, \sigma), \downarrow, \nu'\rangle}{\langle(\sigma_p, \sigma), \downarrow, e; e'\rangle \Downarrow \langle(\sigma''_p, \sigma''), \downarrow, \nu'\rangle}\ E-Seq$$

$$\frac{\langle(\sigma_p, \sigma), \_, e\rangle \Downarrow \langle(\sigma'_p, \sigma'), \lnot, \nu\rangle}{\langle(\sigma_p, \sigma), \_, e; e'\rangle \Downarrow \langle(\sigma'_p, \sigma'), \lnot, \nu\rangle}\ E-Skip$$

$$\frac{\begin{array}{l}\sigma; e_s \Downarrow \nu_s \quad \sigma; a_i \Downarrow \nu_i|_{1 \leqslant i \leqslant n} \quad \Gamma \vdash \nu_s : \mathsf{T}_{\nu_s} \\ \exists \mathsf{T_s}.\ (\mathsf{T_s}, \mathsf{T_r}, \langle(\mathsf{a_i}, \mathsf{T_i})\rangle, e_{body}) \in \zeta(\texttt{f}) \wedge \mathsf{T}_{\nu_s} <: \mathsf{T_s} \wedge \forall \mathsf{T_b}\ \mathsf{T}_{\nu_s} <: \mathsf{T_b} \Rightarrow \mathsf{T_s} <: \mathsf{T_b} \\ \langle \Gamma, \zeta, (\sigma_p, \sigma[\texttt{self} \mapsto \nu_s, a_i \mapsto \nu_i|_{1 \leqslant i \leqslant n}]), \downarrow, e_{body}\rangle \Downarrow \langle(\sigma'_p, \sigma), \lnot, \nu_r\rangle\end{array}}{\langle \Gamma, \zeta, (\sigma_p, \sigma), e_s\texttt{.f(}e_1, .., e_n\texttt{)}\rangle \Downarrow \langle \zeta, (\sigma'_p, \sigma), \downarrow, \nu_r\rangle}\ E-Call$$

$$\frac{\begin{array}{l}\sigma; e_s \Downarrow \nu_s \quad \sigma; a_i \Downarrow \nu_i|_{1 \leqslant i \leqslant n} \quad \Gamma \vdash \nu_s : \mathsf{T}_{\nu_s} \\ \exists \mathsf{T_s}.\ (\mathsf{T_s}, \mathsf{T_r}, \langle(\mathsf{a_i}, \mathsf{T_i})\rangle, e_i, e_p, e_e) \in \xi(\mathsf{m}) \wedge \mathsf{T}_{\nu_s} <: \mathsf{T_s} \wedge \forall \mathsf{T_b}\ \mathsf{T}_{\nu_s} <: \mathsf{T_b} \Rightarrow \mathsf{T_s} <: \mathsf{T_b} \\ \langle \Gamma, \xi, (\sigma_p, \sigma[\texttt{self} \mapsto \nu_s, a_i \mapsto \nu_i|_{1 \leqslant i \leqslant n}]), e_i\rangle \Downarrow \langle(\sigma'_p, \sigma'), \nu_r\rangle \\ \langle \Gamma, \xi, (\sigma'_p, \sigma'[\texttt{result} \mapsto (\varnothing, \varnothing, \varnothing, \mathsf{T_r})]), e_p\rangle \Downarrow \langle(\sigma''_p, \sigma''), \_\rangle \\ \langle \Gamma, \xi(\sigma''_p, \sigma''), e_e\rangle \Downarrow \langle \sigma'''_p, \sigma'''\rangle \quad \sigma'''(\mathit{result}) = (\mathfrak{C}_r, A_r, R_r, \mathsf{T_r})\end{array}}{\langle \Gamma, \xi, (\sigma_p, \sigma), e_s\texttt{.}\underline{\texttt{map}}\ \texttt{m(}e_1, .., e_n\texttt{)}\rangle \Downarrow \langle \xi, (\sigma'''_p, \sigma), (\mathfrak{C}_r, A_r, R_r, \mathsf{T_r})\rangle}\ E-Map$$

$$\frac{\sigma; e \Downarrow \mathsf{True}}{\sigma; e\texttt{.not()} \Downarrow \mathsf{False}}\ \mathsf{E-Not-True} \qquad \frac{\sigma; e \Downarrow \mathsf{False}}{\sigma; e\texttt{.not()} \Downarrow \mathsf{True}}\ \mathsf{E-Not-False}$$

$$\frac{\sigma; e_1 \Downarrow \mathsf{False}}{\sigma; e_1\texttt{.and(}e_2\texttt{))} \Downarrow \mathsf{False}}\ \mathsf{E-And-False} \qquad \frac{\sigma; e_1 \Downarrow \mathsf{True} \qquad \sigma; e_2 \Downarrow \beta}{\sigma; e_1\texttt{.and(}e_2\texttt{)} \Downarrow \beta}\ \mathsf{E-And}$$

$$\frac{\sigma; e_1 \Downarrow \mathsf{True}}{\sigma; e_1\texttt{.or(}e_2\texttt{)} \Downarrow \mathsf{True}}\ \mathsf{E-Or-True} \qquad \frac{\sigma; e_1 \Downarrow \mathsf{False} \qquad \sigma; e_2 \Downarrow \beta}{\sigma; e_1\texttt{.or(}e_2\texttt{)} \Downarrow \beta}\ \mathsf{E-Or}$$

$$\frac{\sigma; e \Downarrow M}{\sigma; e\texttt{.objectsOfType(T)} \Downarrow \{m \in \uplus M | \mathsf{Type}(m) = T\}}\ \mathsf{E-objectsOfType}$$

$$\frac{\sigma; e \Downarrow (\mathfrak{C}, A, R, T)}{\sigma; e\texttt{.rootObjects()} \Downarrow \cup\mathfrak{C}}\ \mathsf{E-rootObjects}$$

$$\frac{\sigma; e \Downarrow (\mathfrak{C}, A, R, T)}{\sigma; e\texttt{.deepclone()} \Downarrow (\mathfrak{C}, A, R, T)}\ \mathsf{E-deepclone}$$

$$\frac{\sigma; e \Downarrow \langle a_1, .., a_n \rangle}{\sigma; e\texttt{->asSet()} \Downarrow \bigcup_{i=1}^{n} \{a_i\}} \; E - asSet - Ord \qquad \frac{\sigma; e \Downarrow A}{\sigma; e\texttt{->asSet()} \Downarrow A} \; E - asSet$$

$$\frac{\sigma; e \Downarrow \{a_1, ..a_n\}|_{a_{i-1} \leqslant a_i}}{\sigma; e\texttt{->asOrderedSet()} \Downarrow \langle a_1, .., a_n \rangle} \; E - asOrderedSet$$

$$\frac{\sigma; e_1 \Downarrow \phi \qquad \sigma; e_2 \Downarrow \mu \qquad \phi(\mu) \neq \bot}{\sigma; e_1\texttt{->hasKey(}e_2\texttt{)} \Downarrow True} \; E - hasKey$$

$$\frac{\sigma; e_1 \Downarrow \phi \qquad \sigma; e_2 \Downarrow \mu \qquad \phi(\mu) = \bot}{\sigma; e_1\texttt{->hasKey(}e_2\texttt{)} \Downarrow False} \; E - hasKey - not$$

$$\frac{\sigma; e \Downarrow \phi \qquad \sigma; e_k \Downarrow \mu \qquad \sigma; e_v \Downarrow \nu}{\sigma; e\texttt{->put(}e_k, e_v\texttt{)} \Downarrow \lambda\, x. \begin{cases} \nu & x = \mu \\ \phi(x) & otherwise \end{cases}} \; E - put$$

$$\frac{\sigma; e \Downarrow \phi \qquad \sigma; e_k \Downarrow \mu}{\sigma; e\texttt{->get(}e_k\texttt{)} \Downarrow \phi(\mu)} \; E - get \qquad \frac{\sigma; e \Downarrow \varnothing}{\sigma; e\texttt{->get(}e_k\texttt{)} \Downarrow \bot} \; E - get - bot$$

$$\frac{\sigma; e \Downarrow \varnothing}{\sigma; e\texttt{->size()} \Downarrow 0} \; E - size - Empty \qquad \frac{\sigma; e \Downarrow \langle \nu_1, .., \nu_n \rangle}{\sigma; e\texttt{->size()} \Downarrow n} \; E - size - Ord$$

$$\frac{\sigma; e \Downarrow A}{\sigma; e\texttt{->size()} \Downarrow |A|} \; E - size - Set \qquad \frac{\sigma; e \Downarrow \langle \nu_1, .., \nu_n \rangle}{\sigma; e\texttt{->sum()} \Downarrow \sum_{i=1}^{n} \nu_i} \; E - sum$$

$$\frac{\sigma; e_1 \Downarrow A \qquad \sigma; e_2 \Downarrow B}{\sigma; e_1\texttt{->union(}e_2\texttt{)} \Downarrow A \cup B} \quad E - Union$$

$$\frac{\sigma; e_1 \Downarrow A \qquad \sigma; e_2 \Downarrow B}{\sigma; e_1\texttt{->intersection(}e_2\texttt{)} \Downarrow A \cap B} \quad E - Intersection$$

$$\frac{\sigma; e \Downarrow \langle \nu_1, .., \nu_n \rangle \qquad \sigma; e_1 \Downarrow i|_{1 \leqslant i \leqslant n} \qquad \sigma; e_2 \Downarrow j|_{i \leqslant j \leqslant n}}{\sigma; e\texttt{->subOrderedSet(}e_1, e_2\texttt{)} \Downarrow \langle \nu_i, \nu_{i+1}, .., \nu_{j-1}, \nu_j \rangle} \quad E - subOrderedSet$$

$$\frac{\sigma; e_1 \Downarrow \langle \nu_1, .. \nu_n \rangle \qquad \sigma; e_2 \Downarrow i|_{1 \leqslant i \leqslant n}}{\sigma; e_1\texttt{->at(}e_2\texttt{)} \Downarrow \nu_i} \quad E - at$$

$$\frac{\sigma; e_1 \Downarrow \langle \nu_1, .. \nu_n \rangle \qquad \sigma; e_2 \Downarrow \nu_i|_{\forall\ j \leqslant i\ \ \nu_j \neq \nu_i}}{\sigma; e_1\texttt{->indexOf(}e_2\texttt{)} \Downarrow i} \quad E - indexOf$$

$$\frac{\sigma; e_1 \Downarrow \varnothing}{\sigma; e_1\texttt{->includes(}e_2\texttt{)} \Downarrow False} \quad E - includes - Empty$$

$$\frac{\sigma; e_1 \Downarrow A \qquad \sigma; e_2 \Downarrow \nu|_{\nu \in A}}{\sigma; e_1\texttt{->includes(}e_2\texttt{)} \Downarrow True} \quad E - includes - True$$

$$\frac{\sigma; e_1 \Downarrow A \qquad \sigma; e_2 \Downarrow \nu|_{\nu \notin A}}{\sigma; e_1\texttt{->includes(}e_2\texttt{)} \Downarrow False} \quad E - includes - False$$

$$\frac{\sigma; e_1 \Downarrow A \qquad \sigma; e_2 \Downarrow \nu}{\sigma; e_1\texttt{->including(}e_2\texttt{)} \Downarrow A \cup \{\nu\}} \quad E - including$$

$$\frac{\sigma; e_1 \Downarrow A \qquad \sigma; e_2 \Downarrow \nu}{\sigma; e_1\texttt{->excluding(}e_2\texttt{)} \Downarrow A - \{\nu\}} \quad E - excluding$$

$$\frac{\sigma; e \Downarrow \varnothing}{\sigma; e\texttt{->isEmpty()} \Downarrow True} \quad E - isEmpty$$

$$\frac{\sigma; e \Downarrow A \neq \varnothing}{\sigma; e\texttt{->isEmpty()} \Downarrow False}\ E - isEmpty - not$$

$$\frac{\sigma; e \Downarrow A \qquad \forall\ a \in A.\ \sigma[x \mapsto a]; e_p \Downarrow \mathit{False}}{\sigma; e\texttt{->exists(x|}e_p\texttt{)} \Downarrow \mathit{False}}\ E-\mathit{exists}-\mathit{not}$$

$$\frac{\sigma; e \Downarrow A \qquad \exists\ a \in A.\ \sigma[x \mapsto a]; e_p \Downarrow \mathit{True}}{\sigma; e\texttt{->exists(x|}e_p\texttt{)} \Downarrow \mathit{True}}\ E-\mathit{exists}$$

$$\frac{\sigma; e \Downarrow \varnothing}{\sigma; e\texttt{->exists(x|}e_p\texttt{)} \Downarrow \mathit{False}}\ E-\mathit{exists}-\mathit{Empty}$$

$$\frac{\sigma; e \Downarrow A \qquad \exists\ a \in A.\ \sigma[x \mapsto a]; e_p \Downarrow \mathit{False}}{\sigma; e\texttt{->forAll(x|}e_p\texttt{)} \Downarrow \mathit{False}}\ E-\mathit{forAll}-\mathit{not}$$

$$\frac{\sigma; e \Downarrow A \qquad \forall\ a \in A.\ \sigma[x \mapsto a]; e_p \Downarrow \mathit{True}}{\sigma; e\texttt{->forAll(x|}e_p\texttt{)} \Downarrow \mathit{True}}\ E-\mathit{forAll}$$

$$\frac{\sigma; e \Downarrow \varnothing}{\sigma; e\texttt{->forAll(x|}e_p\texttt{)} \Downarrow \mathit{True}}\ E-\mathit{forAll}-\mathit{Empty}$$

$$\frac{\sigma; e_s \Downarrow \varnothing}{\sigma; e_s\texttt{->}\underline{\texttt{select}}\texttt{(x|}e_p\texttt{)} \Downarrow \varnothing}\ E-\mathit{select}-\mathit{Empty}$$

$$\frac{\sigma; e_s \Downarrow \{v_1, .., v_n\} \qquad \sigma[x \mapsto v_i]; e_p \Downarrow p_i|_{1 \leqslant i \leqslant n}}{\sigma; e_s\texttt{->}\underline{\texttt{select}}\texttt{(x|}e_p\texttt{)} \Downarrow \{v_i | p_i = \mathit{True}\}}\ E-\mathit{select}$$

$$\frac{\sigma; e_s \Downarrow \varnothing}{\sigma; e_s\texttt{->}\underline{\texttt{collect}}\texttt{(x|}e_c\texttt{)} \Downarrow \varnothing}\ E-\mathit{collect}-\mathit{Empty}$$

$$\frac{\sigma; e_s \Downarrow \langle v_1, .. v_n \rangle \qquad \sigma[x \mapsto v_i]; e \Downarrow c_i|_{1 \leqslant i \leqslant n}}{\sigma; e_s\texttt{->}\underline{\texttt{collect}}\texttt{(x|}e_c\texttt{)} \Downarrow \langle c_1, .., c_n \rangle}\ E-\mathit{collect}$$

Appendix D

QvtMix Implementation

QVT Code

```
1  import qvt.mix.util;



5  modeltype QVT uses qvtoperational::expressions('http://www.eclipse.org/QVT/1.0.0/Operational');
6  modeltype ImpOCL uses ImperativeOCL('http://www.eclipse.org/qvt/1.0/ImperativeOCL');


9  modeltype OCL uses ocl::utilities('http://www.eclipse.org/ocl/1.1.0/OCL');
10 modeltype OCLECORE uses ocl::ecore('http://www.eclipse.org/ocl/1.1.0/OCL');

12 modeltype OCLTYPE uses ocl::types('http://www.eclipse.org/ocl/1.1.0/OCL');
13 modeltype ECORE uses ecore('http://www.eclipse.org/emf/2002/Ecore');
14 modeltype BTA uses 'http://qvt.bta/1.0';
15 modeltype BOOK uses 'http://book/1.0';



20 ////////////////////////////////    qvtMix
21 transformation QvtMix(
22   in inputMetaModel : ECORE,
23   in inputModel : ECORE,
24   in qvt : QVT,
25   in outputModel : ECORE,
26   in outputMetaModel : ECORE,
27   out residue : QVT
28 );



32 ////////////////////////////////////////////////////////////////////////////
33 //
34 // main
35 //
```

```
36 ////////////////////////////////////////////////////////////////////////////

39 main() {
40   var trans := qvt.rootObjects()[OperationalTransformation]->asOrderedSet()->first();
41   trans.bta(bindingTimes, context);



45   var resQvt := trans.mix();
46   log('import qvt.mix.util;');
47   log('modeltype ECORE uses ecore(\'http://www.eclipse.org/emf/2002/Ecore\');');

49   log(resQvt.print(0));

51   log('
52 ////////////////////////// ModelUtils ////////////////////////////////////////////////////
53 intermediate class ObjectPath {
54   path : String;
55 }

57 helper EObject::path() : String {
58   var parents : OrderedSet(Integer) := OrderedSet{};
59   var containers : OrderedSet(Integer) := OrderedSet{};
60   var path := \'\';
61   var obj := self;
62   var parentObj := obj.eContainer();
63   while (parentObj <> null) {
64     var container := obj.eContainingFeature().oclAsType(EReference);
65     var col := Sequence{};
66     getMultiFeature(parentObj, container.name, col);
67     var position := col->indexOf(obj);
68     path := \'/\' + container.name + \'.\' + position.repr() + path;
69     obj := parentObj;
70     parentObj := obj.eContainer();
71   };
72   if path = \'\' then {
73     path := \'/\';
74   }
75   endif;
76   return path;
77 }

79 helper Model::getObject(p : ObjectPath) : EObject {
80   var str := p.path;
81   var obj := self.rootObjects()->asOrderedSet()->first().oclAsType(EObject);
82   str := str.substringAfter(\'/\');
83   while (str <> null) {
84     var segment := str.substringBefore(\'/\');
85     var pos : String;
86     var cont : String;
87     if segment <> null then {
88       pos := segment.substringAfter(\'.\');
89       cont := segment.substringBefore(\'.\');
90     }
91     else {
92       pos := str.substringAfter(\'.\');
93       cont := str.substringBefore(\'.\');
94     }
95     endif;
96     obj := obj.getObject(cont, pos.asInteger());
```

```
    str := str.substringAfter(\'/\');
  };

  return obj;
}

helper ObjectPath::getObject(model : Model) : EObject {
  return model.getObject(self);
}

helper EObject::getObject(cont : String, pos : Integer) : EObject {
  var col := Sequence{};
  getMultiFeature(self, cont, col);
  return col->at(pos).oclAsType(EObject);
}
  ');
}

////////////////////////// Path ////////////////////////////////////////////////

intermediate class ObjectPath {
  path : String;
  rootObject : EObject;
};

constructor ObjectPath::ObjectPath(root : EObject, p : String) {
  path := p;
  rootObject := root;
}


////////////////////////////////////////////////////////////////////////////////
//
// ObjectPath::getObject
//
////////////////////////////////////////////////////////////////////////////////

helper ObjectPath::getObject(model : Model) : EObject {
  return model.getObject(self);
}


////////////////////////////////////////////////////////////////////////////////
//
// EObject::path
//
////////////////////////////////////////////////////////////////////////////////

helper EObject::path() : ObjectPath {
  var parents : OrderedSet(Integer) := OrderedSet{};
  var containers : OrderedSet(Integer) := OrderedSet{};
  var path := '';
  var obj := self;
  var parentObj := obj.eContainer();
  while (parentObj <> null) {
    var container := obj.eContainingFeature().oclAsType(EReference);
    var col := Sequence{};
    getMultiFeature(parentObj, container.name, col);
    var position := col->indexOf(obj);
    path := '/' + container.name + '.' + position.print() + path;
    obj := parentObj;
```

```
158     parentObj := obj.eContainer();
159   };
160   if path = '' then {
161     path := '/';
162   }
163   endif;
164   return new ObjectPath(obj, path);
165 }


168 ////////////////////////////////////////////////////////////////////////////
169 //
170 // Model::getObject
171 //
172 ////////////////////////////////////////////////////////////////////////////

174 helper Model::getObject(path : ObjectPath) : EObject {
175   var str := path.path;
176   var obj := self.rootObjects()->asOrderedSet()->first().oclAsType(EObject);
177   str := str.substringAfter('/');
178   while (str <> null) {
179     var segment := str.substringBefore('/');
180     var pos : String;
181     var cont : String;
182     if segment <> null then {
183       pos := segment.substringAfter('.');
184       cont := segment.substringBefore('.');
185     }
186     else {
187       pos := str.substringAfter('.');
188       cont := str.substringBefore('.');
189     }
190     endif;
191     obj := obj.getObject(cont, pos.asInteger());
192     str := str.substringAfter('/');
193   };

195   return obj;
196 }


199 ////////////////////////////////////////////////////////////////////////////
200 //
201 // ObjectPath::makeExp
202 //
203 ////////////////////////////////////////////////////////////////////////////

205 helper ObjectPath::makeExp() : OCLExpression {
206   return self.path.makeExp();
207 }


210 ////////////////////////////////////////////////////////////////////////////
211 //
212 // EObject::getObject
213 //
214 ////////////////////////////////////////////////////////////////////////////

216 helper EObject::getObject(cont : String, pos : Integer) : EObject {
217   var col := Sequence{};
218   getMultiFeature(self, cont, col);
```

```
  return col->at(pos).oclAsType(EObject);
}


////////////////////////////////////////////////////////////////////////////
//
// memoizeVariable
//
////////////////////////////////////////////////////////////////////////////

helper memoizeVariable(variable : String, path : ObjectPath, val : OclAny) {
  var newpart := object DictLiteralPart {
      key := path.makeExp();
      value := val.makeExp();
  };
  var prop : EAttribute;
  if memoizeTables->hasKey(variable) then {
    prop := memoizeTables->get(variable);
    var dict := prop.eAnnotations.contents->first().oclAsType(DictLiteralExp);
    var parts := dict.part->asOrderedSet();
    parts += newpart;
    dict.part := parts->asSet();
    var annot := object EAnnotation {
      contents := OrderedSet{dict.oclAsType(EObject)};
    };
    setMultiFeature(prop, 'eAnnotations', Sequence{annot});
  }
  else {
    var dict := object DictLiteralExp {
      part := Set{newpart};
    };
    prop := object EAttribute {
      name := variable;
      eType := object DictionaryType {
        keyType := object PrimitiveType {
          name := 'String';
        };
        elementType := newpart.value.eType.oclAsType(EObject);
        name := 'Dict(String, ' +  newpart.value.typeName() + ')';
      };
      eAnnotations := OrderedSet{
        object EAnnotation {
          contents := OrderedSet{dict.oclAsType(EObject)};
        }
      };
    };
  }
  endif;
  memoizeTables->put(variable, prop);
}


////////////////////////////////////////////////////////////////////////////
//
// getCacheType
//
////////////////////////////////////////////////////////////////////////////

helper getCacheType(cache : String) : String {
  if memoizeTables->hasKey(cache) then {
    var table := memoizeTables->get(cache);
```

```
    return table.eType.oclAsType(DictionaryType).name;
  }
  endif;
  return null;
}


////////////////////////////////////////////////////////////////////////////////
//
// EObject::getElementType
//
////////////////////////////////////////////////////////////////////////////////

helper EObject::getElementType() : EObject {
  return null;
}


////////////////////////////////////////////////////////////////////////////////
//
// EClassifier::getElementType
//
////////////////////////////////////////////////////////////////////////////////

helper EClassifier::getElementType() : EObject {
  return self.oclAsType(EObject);
}


////////////////////////////////////////////////////////////////////////////////
//
// CollectionType::getElementType
//
////////////////////////////////////////////////////////////////////////////////

helper CollectionType::getElementType() : EObject {
  return self.elementType;
}


////////////////////////////////////////////////////////////////////////////////
//
// DictionaryType::getElementType
//
////////////////////////////////////////////////////////////////////////////////

helper DictionaryType::getElementType() : EObject {
  return self.elementType;
}



////////////////////////////////////////////////////////////////////////////////
//
// OclAny::typeName
//
////////////////////////////////////////////////////////////////////////////////

helper OclAny::typeName() : String {
  if self.oclIsKindOf(EObject) then {
    return self.oclAsType(EObject).typeName();
```

```
341   }
342   endif;
343   return 'OclAny';
344 }


347 ////////////////////////////////////////////////////////////////////////////////
348 //
349 // Integer::typeName
350 //
351 ////////////////////////////////////////////////////////////////////////////////

353 helper Integer::typeName() : String {
354   return 'Integer';
355 }


358 ////////////////////////////////////////////////////////////////////////////////
359 //
360 // IntegerLiteralExp::typeName
361 //
362 ////////////////////////////////////////////////////////////////////////////////

364 helper IntegerLiteralExp::typeName() : String {
365   return 'Integer';
366 }



370 ////////////////////////////////////////////////////////////////////////////////
371 //
372 // RealLiterealExp::typeName
373 //
374 ////////////////////////////////////////////////////////////////////////////////

376 helper RealLiteralExp::typeName() : String {
377   return 'Real';
378 }


381 ////////////////////////////////////////////////////////////////////////////////
382 //
383 // String::typeName
384 //
385 ////////////////////////////////////////////////////////////////////////////////

387 helper String::typeName() : String {
388   return 'String';
389 }


392 ////////////////////////////////////////////////////////////////////////////////
393 //
394 // StringLiteralExp::typeName
395 //
396 ////////////////////////////////////////////////////////////////////////////////

398 helper StringLiteralExp::typeName() : String {
399   return 'String';
400 }
```

```
403 ////////////////////////////////////////////////////////////////////////////
404 //
405 // EObject::typeName
406 //
407 ////////////////////////////////////////////////////////////////////////////

409 helper EObject::typeName() : String {
410   return self.eClass().name;
411 }


414 ////////////////////////////////////////////////////////////////////////////
415 //
416 // CollectionWrapper::typeName
417 //
418 ////////////////////////////////////////////////////////////////////////////

420 helper CollectionWrapper::typeName() : String {
421   var type : String := self.type;
422   if not self.collection()->isEmpty() then {
423     type := type +  '(' + self.collection()->selectOne(true).typeName() + ')';
424   }
425   endif;
426   return type;
427 }


430 ////////////////////////////////////////////////////////////////////////////
431 //
432 // CollectionLiteralExp::typeName
433 //
434 ////////////////////////////////////////////////////////////////////////////

436 helper CollectionLiteralExp::typeName() : String {
437   var type : String := self.eType.name.substringBefore('(');
438   type := type + '('  + self.part->first().typeName() + ')';
439   return type;
440 }


443 ////////////////////////////////////////////////////////////////////////////
444 //
445 // CollectionItem::typeName
446 //
447 ////////////////////////////////////////////////////////////////////////////

449 helper CollectionItem::typeName() : String {
450   return self.item.typeName();
451 }


454 ////////////////////////////////////////////////////////////////////////////
455 //
456 // ObjectExp::typeName
457 //
458 ////////////////////////////////////////////////////////////////////////////

460 helper ObjectExp::typeName() : String {
461   return self.instantiatedClass.name;
462 }
```

```
464 ////////////////////// ExpSimplify ////////////////////////////////////////////////
466 intermediate class ArithmaticExp extends OCLExpression {
467   left : OCLExpression;
468   right : OCLExpression;
469   op : String;
470 }
473 ////////////////////////////////////////////////////////////////////////////
474 //
475 // OCLExpression::toArithmaticExp
476 //
477 ////////////////////////////////////////////////////////////////////////////
479 helper OCLExpression::toArithmaticExp() : OCLExpression {
480   return self;
481 }
484 ////////////////////////////////////////////////////////////////////////////
485 //
486 // OperationCallExp::toArithmaticExp
487 //
488 ////////////////////////////////////////////////////////////////////////////
490 helper ocl::ecore::OperationCallExp::toArithmaticExp() : OCLExpression {
491   var oper := self.referredOperation.oclAsType(EOperation);
492   var lexp := self.source.oclAsType(OCLExpression);
493   var rexp := self.argument->first().oclAsType(OCLExpression);
494   var res := object ArithmaticExp {
495     left := lexp.toArithmaticExp();
496     right := rexp.toArithmaticExp();
497   };
498   switch {
499     case (oper.name = '+')
500     {
501       res.op := 'ADD';
502     }
503     case (oper.name = '-')
504     {
505       res.op := 'SUB';
506     }
507     case (oper.name = '*')
508     {
509       res.op := 'MUL';
510     }
511     case (oper.name = '/')
512     {
513       res.op := 'DIV';
514     }
515     else
516     {
517       return self;
518     }
520   };
522   return res;
523 }
```

```
526 ////////////////////////////////////////////////////////////////////////////
527 //
528 // OCLExpression::toOperationCall
529 //
530 ////////////////////////////////////////////////////////////////////////////

532 helper ocl::expressions::OCLExpression::toOperationCall() : ocl::expressions::OCLExpression {
533   return self;
534 }


537 ////////////////////////////////////////////////////////////////////////////
538 //
539 // ArithmaticExp::toOperationCall
540 //
541 ////////////////////////////////////////////////////////////////////////////

543 helper ArithmaticExp::toOperationCall() : ocl::expressions::OCLExpression {
544   var lexp := self.left.toOperationCall();
545   var rexp := self.right.toOperationCall();
546   var res := object OperationCallExp {
547     source := lexp;
548   };
549   res.argument += rexp;
550   var opName : String;
551   switch {
552     case (self.op = 'ADD') {opName := '+'}
553     case (self.op = 'SUB') {opName := '-'}
554     case (self.op = 'MUL') {opName := '*'}
555     case (self.op = 'DIV') {opName := '/'}
556   };
557   res.referredOperation := object EOperation {
558     name := opName;
559   }.oclAsType(EObject);
560   return res;
561 }



565 ////////////////////////////////////////////////////////////////////////////
566 //
567 // makeVariableExp
568 //
569 ////////////////////////////////////////////////////////////////////////////

571 helper makeVariableExp(s : String) : VariableExp {
572   return object VariableExp {
573     referredVariable := object Variable {
574       name := s;
575     };
576     name := referredVariable.oclAsType(Variable).name;
577   };
578 }


581 ////////////////////////////////////////////////////////////////////////////
582 //
583 // OCLExpression::simplify
584 //
```

```
585 ////////////////////////////////////////////////////////////////////////////////

587 helper OCLExpression::simplify() : OCLExpression {
588   return self;
589 }


592 ////////////////////////////////////////////////////////////////////////////////
593 //
594 // OCLExpression::isSame
595 //
596 ////////////////////////////////////////////////////////////////////////////////

598 helper OCLExpression::isSame(r : OCLExpression) : Boolean {
599   return self = r;
600 }


603 ////////////////////////////////////////////////////////////////////////////////
604 //
605 // VaraibleExp::isSame
606 //
607 ////////////////////////////////////////////////////////////////////////////////

609 helper VariableExp::isSame(r : OCLExpression) : Boolean {
610   return r.oclIsKindOf(VariableExp) and
611     self.referredVariable.oclAsType(Variable).name = r.oclAsType(VariableExp).referredVariable.oclAsType(Variable).name
612   ;
613 }


616 ////////////////////////////////////////////////////////////////////////////////
617 //
618 // IntegerLiteralExp::isSame
619 //
620 ////////////////////////////////////////////////////////////////////////////////

622 helper IntegerLiteralExp::isSame(r : OCLExpression) : Boolean {
623   return r.oclIsKindOf(IntegerLiteralExp) and
624     (self.integerSymbol = r.oclAsType(IntegerLiteralExp).integerSymbol)
625   ;
626 }


629 ////////////////////////////////////////////////////////////////////////////////
630 //
631 // ArithmaticExp::isSame
632 //
633 ////////////////////////////////////////////////////////////////////////////////

635 helper ArithmaticExp::isSame(r : OCLExpression) : Boolean {
636   if r.oclIsKindOf(ArithmaticExp) then {
637     var rexp := r.oclAsType(ArithmaticExp);
638     switch {
639       case (self.op <> rexp.op)
640       {
641         return false;
642       }
643       case (self.op = 'MUL' or self.op = 'ADD')
644       {
645         return
```

```
646           (
647             (self.left.isSame(rexp.left)) and (self.right.isSame(rexp.right))
648           ) or
649           (
650             (self.left.isSame(rexp.right)) and (self.right.isSame(rexp.left))
651           )
652         ;
653       }
654       else
655       {
656         return (self.left.isSame(rexp.left)) and (self.right.isSame(rexp.right));
657       }
658     };
659   }
660   endif;
661   return false;
662 }


665 ////////////////////////////////////////////////////////////////////////////
666 //
667 // ArithmaticExp::print
668 //
669 ////////////////////////////////////////////////////////////////////////////

671 helper ArithmaticExp::print(tabs : Integer) : String {
672   var code := '';
673   switch {
674     case (self.op = 'ADD')
675     {
676       return '(' + self.left.print(0) + ' + ' + self.right.print(0) + ')';
677     }
678     case (self.op = 'SUB')
679     {
680       return '(' + self.left.print(0) + ' - ' + self.right.print(0) + ')';
681     }
682     case (self.op = 'MUL')
683     {
684       return '(' + self.left.print(0) + ' * ' + self.right.print(0) + ')';
685     }
686     case (self.op = 'DIV')
687     {
688       return '(' + self.left.print(0) + ' / ' + self.right.print(0) + ')';
689     }
690     case (self.op = 'NEG')
691     {
692       return '-' + self.left.print(0);
693     }
694   };

696   return code;
697 }


700 ////////////////////////////////////////////////////////////////////////////
701 //
702 // ArithmaticExp::simplify
703 //
704 ////////////////////////////////////////////////////////////////////////////

706 helper ArithmaticExp::simplify() : OCLExpression {
```

```
var lexp := self.left.simplify();
var rexp := self.right.simplify();
switch {
  case (self.op = 'ADD')
  {
    switch {
      case (lexp.oclIsKindOf(IntegerLiteralExp) and rexp.oclIsKindOf(IntegerLiteralExp))
      {
        return object IntegerLiteralExp {
          integerSymbol :=
            lexp.oclAsType(IntegerLiteralExp).integerSymbol +
            rexp.oclAsType(IntegerLiteralExp).integerSymbol
          ;
        };
      }
      case (rexp.oclIsKindOf(IntegerLiteralExp) and
          rexp.oclAsType(IntegerLiteralExp).integerSymbol = 0)
      {
        return lexp;
      }
      case (lexp.oclIsKindOf(IntegerLiteralExp) and
          lexp.oclAsType(IntegerLiteralExp).integerSymbol = 0)
      {
        return rexp;
      }
      case (lexp.oclIsKindOf(IntegerLiteralExp))
      {
        return object ArithmaticExp {
          left := rexp;
          right := lexp;
          op := 'ADD';
        };
      }
      case (rexp.oclIsKindOf(IntegerLiteralExp) and
        rexp.oclAsType(IntegerLiteralExp).integerSymbol < 0)
      {
        var rval := rexp.oclAsType(IntegerLiteralExp).integerSymbol;
        return object ArithmaticExp {
          left := lexp;
          right := (-rval).makeExp();
          op := 'SUB';
        }.simplify();
      }
      case (lexp.oclIsKindOf(ArithmaticExp) and
          lexp.oclAsType(ArithmaticExp).op = 'ADD')
      {
        return object ArithmaticExp {
          left := lexp.oclAsType(ArithmaticExp).left.simplify();
          right := object ArithmaticExp {
            left := lexp.oclAsType(ArithmaticExp).right;
            right := rexp;
            op := 'ADD';
          }.simplify();
          op := 'ADD';
        }.simplify();
      }
      case (lexp.oclIsKindOf(ArithmaticExp) and
           lexp.oclAsType(ArithmaticExp).op = 'SUB')
      {
        return object ArithmaticExp {
          left := lexp.oclAsType(ArithmaticExp).left.simplify();
```

```
768             right := object ArithmaticExp {
769               left := object ArithmaticExp {
770                 left := lexp.oclAsType(ArithmaticExp).right;
771                 op := 'NEG';
772               }.simplify();
773               right := rexp;
774               op := 'ADD';
775             };
776             op := 'ADD';
777           }.simplify();
778         }
779         case (lexp.oclIsKindOf(VariableExp) and
780             rexp.oclIsKindOf(ArithmaticExp)) // x + (y + x)
781         {
782           var r := rexp.oclAsType(ArithmaticExp);
783           if (lexp.isSame(r.left)) then { // x + (x + y)
784             return object ArithmaticExp {
785               left := object ArithmaticExp {
786                 left := 2.makeExp();
787                 right := r.left;
788                 op := 'MUL';
789               };
790               right := r.right;
791               op := 'ADD';
792             };
793           }
794           endif;
795           if (lexp.isSame(r.right)) then { // x + (y + x)
796             return object ArithmaticExp {
797               left := object ArithmaticExp {
798                 left := 2.makeExp();
799                 right := r.right;
800                 op := 'MUL';
801               };
802               right := r.left;
803               op := 'ADD';
804             };
805           }
806           endif;
807         }
808         case (lexp.isSame(rexp))
809         {
810           return object ArithmaticExp {
811             left := 2.makeExp();
812             right := rexp;
813             op := 'MUL';
814           };
815         }
816         case ((lexp.oclIsKindOf(ArithmaticExp) and lexp.oclAsType(ArithmaticExp).op = 'MUL') or
817           (rexp.oclIsKindOf(ArithmaticExp) and rexp.oclAsType(ArithmaticExp).op = 'MUL'))
818         {
819           var l : ArithmaticExp;
820           if (lexp.oclIsKindOf(ArithmaticExp)) then {
821             l := lexp.oclAsType(ArithmaticExp);
822           }
823           else {
824             l := object ArithmaticExp {
825               left := 1.makeExp();
826               right := lexp;
827               op := 'MUL';
828             };
```

```
829         }
830         endif;
831         var r : ArithmaticExp;
832         if (rexp.oclIsKindOf(ArithmaticExp)) then {
833           r := rexp.oclAsType(ArithmaticExp);
834         }
835         else {
836           r := object ArithmaticExp {
837             left  := 1.makeExp();
838             right := rexp;
839             op := 'MUL';
840           };
841         }
842         endif;
843         var factor : OCLExpression;
844         var lpart : OCLExpression;
845         var rpart : OCLExpression;

847         if l.op = 'MUL' and r.op = 'MUL' then {
848           switch {
849             case (l.left.isSame(r.left))
850             {
851               factor := l.left.simplify();
852               lpart := l.right.simplify();
853               rpart := r.right.simplify();
854             }
855             case (l.right.isSame(r.right))
856             {
857               factor := l.right.simplify();
858               lpart := l.left.simplify();
859               rpart := r.left.simplify();
860             }
861             case (l.left.isSame(r.right))
862             {
863               factor := l.left.simplify();
864               lpart := l.right.simplify();
865               rpart := r.left.simplify();
866             }
867             else
868             {
869               return self;
870             }
871           }
872         }
873         else {
874           return object ArithmaticExp {
875             left := object ArithmaticExp {
876               left := l;
877               right := r.left;
878               op := 'ADD';
879             }.simplify();
880             right := r.right;
881             op := r.op;
882           }.simplify();
883         }
884         endif;
885         return object ArithmaticExp {
886           left := factor.simplify();
887           right := object ArithmaticExp {
888             left := lpart;
889             right := rpart;
```

```
890             op := 'ADD';
891           }.simplify();
892           op := 'MUL';
893         }.simplify();
894       }
895     };
896   }
897   case (self.op = 'SUB')
898   {
899     switch {
900       case (lexp.oclIsKindOf(IntegerLiteralExp) and
901           rexp.oclIsKindOf(IntegerLiteralExp))
902       {
903         return object IntegerLiteralExp {
904           integerSymbol :=
905             lexp.oclAsType(IntegerLiteralExp).integerSymbol -
906             rexp.oclAsType(IntegerLiteralExp).integerSymbol
907           ;
908         };
909       }
910       case (rexp.oclIsKindOf(IntegerLiteralExp) and
911           rexp.oclAsType(IntegerLiteralExp).integerSymbol = 0)
912       {
913         return lexp;
914       }
915       case (lexp.oclIsKindOf(IntegerLiteralExp) and
916           lexp.oclAsType(IntegerLiteralExp).integerSymbol = 0)
917       {
918         return object ArithmaticExp {
919           left := rexp;
920           right := null;
921           op := 'NEG';
922         }.simplify();
923       }
924       case (lexp.isSame(rexp))
925       {
926         return 0.makeExp();
927       }
928       case (rexp.oclIsKindOf(IntegerLiteralExp))
929       {
930         var rval := rexp.oclAsType(IntegerLiteralExp).integerSymbol;
931         if rval < 0 then {
932           return object ArithmaticExp {
933             left := lexp;
934             right := (-rval).makeExp();
935             op := 'ADD';
936           };
937         }
938         endif;
939         return object ArithmaticExp {
940           left := lexp;
941           right := rexp;
942           op := 'SUB';
943         };
944       }

946     }
947   }
948   case (self.op = 'MUL')
949   {
950     switch {
```

```
951         case ((rexp.oclIsKindOf(IntegerLiteralExp) and
952             rexp.oclAsType(IntegerLiteralExp).integerSymbol = 0) or
953             (lexp.oclIsKindOf(IntegerLiteralExp) and
954             lexp.oclAsType(IntegerLiteralExp).integerSymbol = 0))
955         {
956           return object IntegerLiteralExp {
957             integerSymbol := 0;
958           };
959         }
960         case (rexp.oclIsKindOf(IntegerLiteralExp) and
961             rexp.oclAsType(IntegerLiteralExp).integerSymbol = 1)
962         {
963           return lexp;
964         }
965         case (lexp.oclIsKindOf(IntegerLiteralExp) and
966             lexp.oclAsType(IntegerLiteralExp).integerSymbol = 1)
967         {
968           return rexp;
969         }
970         case (lexp.oclIsKindOf(IntegerLiteralExp) and
971             rexp.oclIsKindOf(IntegerLiteralExp))
972         {
973           return object IntegerLiteralExp {
974             integerSymbol :=
975               lexp.oclAsType(IntegerLiteralExp).integerSymbol *
976               rexp.oclAsType(IntegerLiteralExp).integerSymbol
977             ;
978           };
979         }
980         case (rexp.oclIsKindOf(IntegerLiteralExp))
981         {
982           return object ArithmaticExp {
983             left := rexp;
984             right := lexp;
985             op := 'MUL';
986           }.simplify();
987         }
988         case (lexp.oclIsKindOf(ArithmaticExp) and
989             lexp.oclAsType(ArithmaticExp).op = 'MUL')
990         {
991           return object ArithmaticExp {
992             left := lexp.oclAsType(ArithmaticExp).left.simplify();
993             right := object ArithmaticExp {
994               left := lexp.oclAsType(ArithmaticExp).right;
995               right := rexp;
996               op := 'MUL';
997             }.simplify();
998             op := 'MUL';
999           }.simplify();
1000        }
1001        case (lexp.oclIsKindOf(VariableExp) and
1002            rexp.oclIsKindOf(ArithmaticExp))
1003        {
1004          var l := object ArithmaticExp {
1005            left := lexp;
1006            right := rexp;
1007            op := 'MUL';
1008          };
1009          if (rexp.oclIsKindOf(ArithmaticExp)) then {
1010            var r := rexp.oclAsType(ArithmaticExp);
1011            l.right := r.left;
```

```
            return object ArithmaticExp {
              left := l.simplify();
              right := r.right;
              op := 'MUL';
            }.simplify();
          }
          else {
            return l.simplify();
          }
          endif;
        }

      };
    }
    case (self.op = 'DIV')
    {
      switch {
        case (rexp.oclIsKindOf(IntegerLiteralExp) and
            rexp.oclAsType(IntegerLiteralExp).integerSymbol = 0)
        {
          return null;
        }
        case (lexp.oclIsKindOf(IntegerLiteralExp) and
            lexp.oclAsType(IntegerLiteralExp).integerSymbol = 0)
        {
          return lexp;
        }
        case (lexp.isSame(rexp))
        {
          return 1.makeExp();
        }
        case (rexp.oclIsKindOf(IntegerLiteralExp) and
            lexp.oclIsKindOf(IntegerLiteralExp))
        {
          return self;
        }
        case (lexp.oclIsKindOf(ArithmaticExp) and
            lexp.oclAsType(ArithmaticExp).op = 'MUL')
        {
          var l := lexp.oclAsType(ArithmaticExp);
          return object ArithmaticExp {
            left := l.left;
            right := object ArithmaticExp {
              left := l.right;
              right := rexp;
              op := 'DIV';
            };
            op := 'MUL';
          }.simplify();
        }
      };
    }
    case (self.op = 'NEG')
    {
      switch {
        case (lexp.oclIsKindOf(IntegerLiteralExp)) {
          return (-(lexp.oclAsType(IntegerLiteralExp).integerSymbol)).makeExp();
        }
        case (lexp.oclIsKindOf(ArithmaticExp))
        {
          var l := lexp.oclAsType(ArithmaticExp);
```

```
          switch {
              case (l.op = 'NEG')
              {
                return l.left.simplify();
              }
              case (l.op = 'ADD')
              {
                return object ArithmaticExp {
                  left := object ArithmaticExp {
                    left := l.left.simplify();
                    op := 'NEG';
                  };
                  right := object ArithmaticExp {
                    left := l.right.simplify();
                    op := 'NEG';
                  };
                  op := 'SUB';
                }.simplify();
              }
              case (l.op = 'SUB')
              {
                return object ArithmaticExp {
                  left := object ArithmaticExp {
                    left := l.left.simplify();
                    op := 'NEG';
                  };
                  right := object ArithmaticExp {
                    left := l.right.simplify();
                    op := 'NEG';
                  };
                  op := 'ADD';
                }.simplify();
              }
          };
        }
      };
    }
  };
  return object ArithmaticExp {
    left := lexp;
    right := rexp;
    op := self.op
  };
}
```

```
////////////////////// QvtEval ////////////////////////////////////////////////////////
property helpers : OrderedSet(Helper) = null;
property mappings : OrderedSet(MappingOperation) = null;
property properties : OrderedSet(EAttribute) = null;
property configProperties : OrderedSet(EAttribute) = null;
property usedModelTypes : OrderedSet(ModelType) = null;
property mappingCallLevel : Integer = 0;

////////////////////////////////////////////////////////////////////////////////////////////
//
//  Classes
//
```

```
1134 ////////////////////////////////////////////////////////////////////////////////////////////////////

1136 intermediate class Environment {
1137   localEnv : Dict(String, Frame);
1138   parentEnv : Environment;
1139 }

1141 constructor Environment::Environment() {
1142   localEnv := Dict{};
1143   parentEnv := null;
1144 }


1147 ////////////////////////////////////////////////////////////////////////////
1148 //
1149 // createEnvironment
1150 //
1151 ////////////////////////////////////////////////////////////////////////////

1153 helper createEnvironment() : Environment {
1154   return new Environment();
1155 }

1157 constructor Environment::Environment(parent : Environment) {
1158   localEnv := Dict {};
1159   parentEnv := parent;
1160 }


1163 ////////////////////////////////////////////////////////////////////////////
1164 //
1165 // createEnvironment
1166 //
1167 ////////////////////////////////////////////////////////////////////////////

1169 helper createEnvironment(parent : Environment) : Environment {
1170   return new Environment(parent);
1171 }

1173 constructor Environment::Environment(name : String, val : OclAny, parent : Environment ) {
1174   parentEnv := parent;
1175   localEnv := Dict {};
1176   localEnv->put(name, createFrame(val));
1177 }


1180 ////////////////////////////////////////////////////////////////////////////
1181 //
1182 // createEnvironment
1183 //
1184 ////////////////////////////////////////////////////////////////////////////

1186 helper createEnvironment(name : String, val : OclAny, parent : Environment) : Environment {
1187   return new Environment(name, val, parent);
1188 }


1191 ////////////////////////////////////////////////////////////////////////////
1192 //
1193 // Environment::hasKey
1194 //
```

```
1195 ////////////////////////////////////////////////////////////////////////////

1197 helper Environment::hasKey(name : String) : Boolean {
1198   if (self.localEnv->hasKey(name)) then {
1199     return true;
1200   }
1201   endif;
1202   return self.parentEnv.hasKey(name);
1203 }


1206 ////////////////////////////////////////////////////////////////////////////
1207 //
1208 // Environment::put
1209 //
1210 ////////////////////////////////////////////////////////////////////////////

1212 helper Environment::put(name : String, val : Frame) : Environment {
1213   if  (not self.localEnv->hasKey(name)) and self.parentEnv.hasKey(name) then {
1214       self.parentEnv->put(name, val);
1215   }
1216   else {
1217       self.localEnv->put(name, val);
1218   }
1219   endif;
1220   return self;
1221 }


1224 ////////////////////////////////////////////////////////////////////////////
1225 //
1226 // Environment::get
1227 //
1228 ////////////////////////////////////////////////////////////////////////////

1230 helper Environment::get(name : String) : Frame{
1231   if self.localEnv->hasKey(name) then {
1232     return self.localEnv->get(name);
1233   }
1234   endif;
1235   return self.parentEnv.get(name);
1236 }


1239 ////////////////////////////////////////////////////////////////////////////
1240 //
1241 // getEnvironment
1242 //
1243 ////////////////////////////////////////////////////////////////////////////

1245 helper getEnvironment(e : OclAny) : Environment {
1246   if (e.oclIsTypeOf(Environment)) then {
1247     return e.oclAsType(Environment);
1248   }
1249   endif;
1250   return null;
1251 }


1254 ////////////////////////////////////////////////////////////////////////////
1255 //
```

```
// Environment::copy
//
////////////////////////////////////////////////////////////////////////////////

helper Environment::copy() : Environment {
  var env : Environment := new Environment();
  var local := self.localEnv;
  var goUp := true;
  while (goUp) {
    local->keys()->forEach(k) {
      env.put(k, local->get(k));
    };
    if (self.parentEnv <> null) then {
      env := object Environment {
        parentEnv := env;
      };
      local := self.parentEnv.localEnv;
    }
    else {
      break;
    }
    endif;
  };

  return env;
}

intermediate class Function {};


////////////////////////////////////////////////////////////////////////////////
//
// Function::func
//
////////////////////////////////////////////////////////////////////////////////

helper Function::func(arg : OclAny) : OclAny {
  return null;
}

intermediate class Eq extends Function {};

////////////////////////////////////////////////////////////////////////////////
//
// Eq::func
//
////////////////////////////////////////////////////////////////////////////////

helper Eq::func(arg0 : OclAny, arg1 : OclAny) : OclAny {
  return arg0 = arg1;
}


////////////////////////////////////////////////////////////////////////////////
//
// Function::func
//
////////////////////////////////////////////////////////////////////////////////

helper Function::func(arg0 : OclAny, arg1 : OclAny) : OclAny {
  return null;
```

```
1317 }


1320 ////////////////////////////////////////////////////////////////////////////////
1321 //
1322 // ImperativeIterateExp::makeCondition
1323 //
1324 ////////////////////////////////////////////////////////////////////////////////

1326 helper ImperativeIterateExp::makeCondition(env : Environment) : Function {
1327   var opCall := self.condition.oclAsType(OperationCallExp);
1328   var opName := opCall.referredOperation.oclAsType(EOperation).name;
1329   var cond : Function := null;
1330   if opName = '=' then {
1331     cond := new Eq();
1332   }
1333   endif;
1334   return cond;
1335 }


1338 ////////////////////////////////////////////////////////////////////////////////
1339 //
1340 // ImperativeIterateExp::makeCollector
1341 //
1342 ////////////////////////////////////////////////////////////////////////////////

1344 helper ImperativeIterateExp::makeCollector(env : Environment) : Function {
1345   return null;
1346 }


1349 intermediate class CollectionWrapper extends  EObject {
1350   set : Set(OclAny);
1351   seq : Sequence(OclAny);
1352   ord : OrderedSet(OclAny);
1353   list : List(OclAny);
1354   bag : Bag(OclAny);
1355   type : String;
1356 }

1358 constructor CollectionWrapper::CollectionWrapper(c : Sequence(OclAny)) {
1359   type := 'Sequence';
1360   seq := c;
1361 }

1363 constructor CollectionWrapper::CollectionWrapper(c : OrderedSet(OclAny)) {
1364   type := 'OrderedSet';
1365   ord := c;
1366 }

1368 constructor CollectionWrapper::CollectionWrapper(c : Set(OclAny)) {
1369   type := 'Set';
1370   set := c;
1371 }

1373 constructor CollectionWrapper::CollectionWrapper(c : List(OclAny)) {
1374   type := 'List';
1375   list := c;
1376 }
```

```
1378 constructor CollectionWrapper::CollectionWrapper(c : Bag(OclAny)) {
1379   type := 'Bag';
1380   bag := c;
1381 }

1383 constructor CollectionWrapper::CollectionWrapper(c : Collection(OclAny)) {
1384   type := 'Collection';
1385 }


1388 ////////////////////////////////////////////////////////////////////////////
1389 //
1390 // CollectionWrapper
1391 //
1392 ////////////////////////////////////////////////////////////////////////////

1394 helper CollectionWrapper::collection() : Collection(OclAny) {
1395   switch {
1396     case (self.type = 'Bag')
1397       {return self.bag}
1398     case (self.type = 'List')
1399       {return self.list}
1400     case (self.type = 'OrderedSet')
1401       {return self.ord}
1402     case (self.type = 'Sequence')
1403       {return self.seq}
1404     case (self.type = 'Set')
1405       {return self.set}
1406   };
1407   return null;
1408 }


1411 ////////////////////////////////////////////////////////////////////////////
1412 //
1413 // createCollectionWrapper
1414 //
1415 ////////////////////////////////////////////////////////////////////////////

1417 helper createCollectionWrapper(c : Bag(OclAny)) : CollectionWrapper {
1418   return new CollectionWrapper(c);
1419 }


1422 ////////////////////////////////////////////////////////////////////////////
1423 //
1424 // createCollectionWrapper
1425 //
1426 ////////////////////////////////////////////////////////////////////////////

1428 helper createCollectionWrapper(c : Set(OclAny)) : CollectionWrapper {
1429   return new CollectionWrapper(c);
1430 }


1433 ////////////////////////////////////////////////////////////////////////////
1434 //
1435 // createCollectionWrapper
1436 //
1437 ////////////////////////////////////////////////////////////////////////////
```

```
1439 helper createCollectionWrapper(c : OrderedSet(OclAny)) : CollectionWrapper {
1440   return new CollectionWrapper(c);
1441 }


1444 ////////////////////////////////////////////////////////////////////////////
1445 //
1446 // createCollectionWrapper
1447 //
1448 ////////////////////////////////////////////////////////////////////////////

1450 helper createCollectionWrapper(c : Sequence(OclAny)) : CollectionWrapper {
1451   return new CollectionWrapper(c);
1452 }


1455 ////////////////////////////////////////////////////////////////////////////
1456 //
1457 // createCollectionWrapper
1458 //
1459 ////////////////////////////////////////////////////////////////////////////

1461 helper createCollectionWrapper(c : List(OclAny)) : CollectionWrapper {
1462   return new CollectionWrapper(c);
1463 }


1466 ////////////////////////////////////////////////////////////////////////////
1467 //
1468 // getWrappedCollection
1469 //
1470 ////////////////////////////////////////////////////////////////////////////

1472 helper getWrappedCollection(c : OclAny) : Collection(OclAny) {
1473   if (c.oclIsTypeOf(CollectionWrapper)) then {
1474     return c.oclAsType(CollectionWrapper).collection();
1475   }
1476   endif;
1477   return null;
1478 }


1481 ////////////////////////////////////////////////////////////////////////////
1482 //
1483 // getCollectionWrapper
1484 //
1485 ////////////////////////////////////////////////////////////////////////////

1487 helper getCollectionWrapper(c : OclAny) : CollectionWrapper {
1488   if (c.oclIsTypeOf(CollectionWrapper)) then {
1489     return c.oclAsType(CollectionWrapper);
1490   }
1491   endif;
1492   return null;
1493 }


1496 ////////////////////////////////////////////////////////////////////////////
1497 //
1498 // setCollectionWrapper
1499 //
```

```
1500 ////////////////////////////////////////////////////////////////////////////
1501
1502 helper setCollectionWrapper(inout c : EObject, col : OrderedSet(OclAny)) {
1503   if (c.oclIsTypeOf(CollectionWrapper)) then {
1504     c.oclAsType(CollectionWrapper).ord := col;
1505   }
1506   endif;
1507
1508 }
1509
1510
1511 ////////////////////////////////////////////////////////////////////////////
1512 //
1513 // OclAny::asSet
1514 //
1515 ////////////////////////////////////////////////////////////////////////////
1516
1517 helper OclAny::asSet() : Set(OclAny) {
1518   if self.oclIsKindOf(CollectionWrapper)then {
1519     self.oclAsType(CollectionWrapper).asSet();
1520   }
1521   endif;
1522   return Set{self};
1523 }
1524
1525
1526 ////////////////////////////////////////////////////////////////////////////
1527 //
1528 // CollectionWrapper::asSet
1529 //
1530 ////////////////////////////////////////////////////////////////////////////
1531
1532 helper CollectionWrapper::asSet() : Set(OclAny) {
1533   switch {
1534     case (self.type = 'Bag')
1535       {return self.bag->asSet()}
1536     case (self.type = 'List')
1537       {
1538         var res := Sequence{};
1539         self.list->forEach(element) {
1540           res->append(element)
1541         };
1542         return res->asSet();
1543       }
1544     case (self.type = 'OrderedSet')
1545       {return self.ord->asSet()}
1546     case (self.type = 'Sequence')
1547       {return self.seq->asSet()}
1548     case (self.type = 'Set')
1549       {return self.set}
1550   };
1551   return null;
1552 }
1553
1554
1555 ////////////////////////////////////////////////////////////////////////////
1556 //
1557 // CollectionWrapper::asOrderedSet
1558 //
1559 ////////////////////////////////////////////////////////////////////////////
```

```
1561 helper CollectionWrapper::asOrderedSet() : OrderedSet(OclAny) {
1562   switch {
1563     case (self.type = 'Bag')
1564       {return self.bag->asOrderedSet()}
1565     case (self.type = 'List')
1566       {
1567         var res := OrderedSet{};
1568         self.list->forEach(element) {
1569           res->append(element)
1570         };
1571         return res;
1572       }
1573     case (self.type = 'OrderedSet')
1574       {return self.ord}
1575     case (self.type = 'Sequence')
1576       {return self.seq->asOrderedSet()}
1577     case (self.type = 'Set')
1578       {return self.set->asOrderedSet()}
1579   };
1580   return null;
1581 }

1583 ////////////////////////////////////////////////////////////////////////////
1584 //
1585 // CollectionWrapper::asBag
1586 //
1587 ////////////////////////////////////////////////////////////////////////////

1589 helper CollectionWrapper::asBag() : Bag(OclAny) {
1590   switch {
1591     case (self.type = 'Bag')
1592       {return self.bag}
1593     case (self.type = 'List')
1594       {
1595         var res := OrderedSet{};
1596         self.list->forEach(element) {
1597           res->append(element)
1598         };
1599         return res->asBag();
1600       }
1601     case (self.type = 'OrderedSet')
1602       {return self.ord->asBag()}
1603     case (self.type = 'Sequence')
1604       {return self.seq->asBag()}
1605     case (self.type = 'Set')
1606       {return self.set->asBag()}
1607   };
1608   return null;
1609 }


1612 ////////////////////////////////////////////////////////////////////////////
1613 //
1614 // CollectionWrapper::asList
1615 //
1616 ////////////////////////////////////////////////////////////////////////////

1618 helper CollectionWrapper::asList() : List(OclAny) {
1619   switch {
1620     case (self.type = 'Bag')
1621       {return self.bag->asList()}
```

```
    case (self.type = 'List')
      {return self.list}
    case (self.type = 'OrderedSet')
      {return self.ord->asList()}
    case (self.type = 'Sequence')
      {return self.seq->asList()}
    case (self.type = 'Set')
      {return self.set->asList()}
  };
  return null;
}



////////////////////////////////////////////////////////////////////////////
//
// CollectonWrapper::print
//
////////////////////////////////////////////////////////////////////////////

helper CollectionWrapper::print() : String {
  var output : String =  '[';
  if (self.collection()->notEmpty()) then {
    var firstObj := self.collection()->any(true);
    output := output + firstObj.repr();
    //self.collection->
    self.collection()->forEach(e| e <> firstObj){
      output := output + ', ' + e.repr();
    };
  }
  endif;
  return output + ']';
}

intermediate class FrameFactory {
  static instance : FrameFactory;
};



////////////////////////////////////////////////////////////////////////////
//
// creteFrame
//
////////////////////////////////////////////////////////////////////////////

helper createFrame(val : OclAny) : Frame {
  switch {
    case (val.oclIsKindOf(EObject))
      {return new ObjectFrame(val.oclAsType(EObject))}
    case (val.oclIsKindOf(Integer))
      {return new IntegerFrame(val.oclAsType(Integer))}
    case (val.oclIsKindOf(String))
      {return new StringFrame(val.oclAsType(String))}
    case (val.oclIsKindOf(Boolean))
      {return new BooleanFrame(val.oclAsType(Boolean))}
    else {return new Frame(val)}
  };
  return null;
}
```

```
1684 ////////////////////////////////////////////////////////////////////////////////
1685 //
1686 // updateFrame
1687 //
1688 ////////////////////////////////////////////////////////////////////////////////


1691 helper updateFrame(frame : ObjectFrame, val : EObject) : Frame {
1692   return new ObjectFrame(val);
1693 }


1696 ////////////////////////////////////////////////////////////////////////////////
1697 //
1698 // updateFrame
1699 //
1700 ////////////////////////////////////////////////////////////////////////////////

1702 helper updateFrame(frame : IntegerFrame, val : Integer) : Frame {
1703   return new IntegerFrame(val);
1704 }

1706 intermediate class Frame {
1707   type : AnyType;
1708   scope : Integer;
1709   binding : String;
1710   anyValue : OclAny;
1711 }

1713 constructor Frame::Frame(val : OclAny) {
1714   anyValue := val;
1715 }

1717 ////////////////////////////////////////////////////////////////////////////////
1718 //
1719 // Frame::value
1720 //
1721 ////////////////////////////////////////////////////////////////////////////////

1723 helper Frame::value() : OclAny {
1724   return self.anyValue;
1725 }

1727 intermediate class ObjectFrame extends Frame{
1728   objValue : EObject;
1729 }
1730 constructor ObjectFrame::ObjectFrame(val : EObject) {
1731   objValue := val;
1732 }


1735 ////////////////////////////////////////////////////////////////////////////////
1736 //
1737 // ObjectFrame::value
1738 //
1739 ////////////////////////////////////////////////////////////////////////////////

1741 helper ObjectFrame::value() : OclAny {
1742   return self.objValue;
1743 }
```

```
1745 intermediate class IntegerFrame extends Frame {
1746   intValue : Integer;
1747 }

1749 constructor IntegerFrame::IntegerFrame(val : Integer) {
1750   intValue := val;
1751 }


1754 ////////////////////////////////////////////////////////////////////////////
1755 //
1756 // IntegerFrame::value
1757 //
1758 ////////////////////////////////////////////////////////////////////////////

1760 helper IntegerFrame::value() : OclAny {
1761   return self.intValue;
1762 }

1764 intermediate class StringFrame extends Frame {
1765   strValue : String;
1766 }

1768 constructor StringFrame::StringFrame(val : String) {
1769   strValue := val;
1770 }


1773 ////////////////////////////////////////////////////////////////////////////
1774 //
1775 // StringFrame::value
1776 //
1777 ////////////////////////////////////////////////////////////////////////////

1779 helper StringFrame::value() : OclAny {
1780   return self.strValue;
1781 }

1783 intermediate class BooleanFrame extends Frame {
1784   boolValue : Boolean;
1785 }

1787 constructor BooleanFrame::BooleanFrame(val : Boolean) {
1788   boolValue := val;
1789 }


1792 ////////////////////////////////////////////////////////////////////////////
1793 //
1794 // BooleanFrame::value
1795 //
1796 ////////////////////////////////////////////////////////////////////////////

1798 helper BooleanFrame::value() : OclAny {
1799   return self.boolValue;
1800 }


1803 ////////////////////////////////////////////////////////////////////////////
1804 //
```

```
1805 // OclAny::invoke
1806 //
1807 ////////////////////////////////////////////////////////////////////////////////

1809 helper OclAny::invoke(op : EOperation, args : Sequence(OclAny)) : OclAny {
1810   var opName := op.name;
1811   if (not self.oclIsKindOf(CollectionWrapper)) then
1812     switch {
1813       case (opName = '=') {return self = args->first();}
1814       case (opName = '<>') {return self <> args->first();}
1815       case (opName = 'asSequence') {return new CollectionWrapper(self->asSequence());}
1816       case (opName = 'asSet') {return new CollectionWrapper(self->asSet());}
1817       case (opName = 'asBag') {return new CollectionWrapper(self->asBag());}
1818       case (opName = 'asOrderedSet') {return new CollectionWrapper(self->asOrderedSet());}
1819       case (opName = 'asList') {return new CollectionWrapper(self->asList());}

1821     }
1822   endif;
1823   var builtin := self.invoke(opName, args);
1824   if builtin = null then
1825     return nonBuiltinOperationInvoke(self, op, args)
1826   endif;
1827   return builtin;
1828 }


1831 ////////////////////////////////////////////////////////////////////////////////
1832 //
1833 // OclAny::invoke
1834 //
1835 ////////////////////////////////////////////////////////////////////////////////

1837 helper OclAny::invoke(opName : String, args : Sequence(OclAny)) : OclAny {
1838   if self.oclIsKindOf(EObject) then
1839     return self.oclAsType(EObject).invoke(opName, args)
1840   endif;
1841   return null;
1842 }


1845 ////////////////////////////////////////////////////////////////////////////////
1846 //
1847 // nonBuiltinOperationInvoke
1848 //
1849 ////////////////////////////////////////////////////////////////////////////////

1851 helper nonBuiltinOperationInvoke(source : OclAny, op : EOperation, args : Sequence(OclAny)) : OclAny {
1852   // look-up defined operations
1853   // evaluate the operation body for the source and args
1854   var srcObj := source.oclAsType(EObject);
1855   var srcClass := srcObj.eClass();
1856   var res := srcObj.eInvoke(op, args->oclAsType(EObject));
1857   return null;
1858 }


1861 ////////////////////////////////////////////////////////////////////////////////
1862 //
1863 // String::invoke
1864 //
1865 ////////////////////////////////////////////////////////////////////////////////
```

```
1867 helper String::invoke(opName : String, args : Sequence(OclAny)) : OclAny {
1868   switch {
1869     case (opName = '=')
1870       {return self = args->first().oclAsType(String)}
1871     case (opName = '<>')
1872       {return self <> args->first().oclAsType(String)}
1873     case (opName = 'size')
1874       {return self.size()}
1875     case (opName = 'concat')
1876       {return self.concat(args->first().oclAsType(String))}
1877     case (opName = 'substring')
1878       {return self.substring(args->first().oclAsType(Integer), args->at(2).oclAsType(Integer))}
1879     case (opName = 'toInteger')
1880       {return self.toInteger()}
1881     case (opName = 'toReal')
1882       {return self.toReal()}
1883     case (opName = 'toLower')
1884       {return self.toLower()}
1885     case (opName = 'toUpper')
1886       {return self.toUpper()}
1887     case (opName = '+')
1888       {return self + args->first().oclAsType(String)}
1889     case (opName = 'addSuffixNumber')
1890       {return self.addSuffixNumber()}
1891     case (opName = 'asBoolean')
1892       {return self.asBoolean()}
1893     case (opName = 'asFloat')
1894       {return self.asFloat()}
1895     case (opName = 'asInteger')
1896       {return self.asInteger()}
1897     case (opName = 'endsWith')
1898       {return self.endsWith(args->first().oclAsType(String))}
1899     case (opName = 'equalsIgnoreCase')
1900       {return self.equalsIgnoreCase(args->first().oclAsType(String))}
1901     case (opName = 'find')
1902       {return self.find(args->first().oclAsType(String))}
1903     case (opName = 'firstToUpper')
1904       {return self.firstToUpper()}
1905     case (opName = 'format')
1906       {
1907         log("format not supported yet");
1908         assert fatal (true);
1909       }
1910     case (opName = 'getStrCounter')
1911       {return self.getStrCounter(args->first().oclAsType(String))}
1912     case (opName = 'incrStrCounter')
1913       {return self.incrStrCounter(args->first().oclAsType(String))}
1914     case (opName = 'indexOf')
1915       {return self.indexOf(args->first().oclAsType(String))}
1916     case (opName = 'isQuoted')
1917       {return self.isQuoted(args->first().oclAsType(String))}
1918     case (opName = 'lastToUpper')
1919       {return self.lastToUpper()}
1920     case (opName = 'length')
1921       {return self.length()}
1922     case (opName = 'match')
1923       {return self.match(args->first().oclAsType(String))}
1924     case (opName = 'matchBoolean')
1925       {return self.matchBoolean(args->first().oclAsType(Boolean))}
1926     case (opName = 'matchFloat')
```

```
1927       {return self.matchFloat(args->first().oclAsType(Real))}
1928     case (opName = 'matchIdentifier')
1929       {return self.matchIdentifier(args->first().oclAsType(String))}
1930     case (opName = 'matchInteger')
1931       {return self.matchInteger(args->first().oclAsType(Integer))}
1932     case (opName = 'normalizeSpace')
1933       {return self.normalizeSpace()}
1934     case (opName = 'replace')
1935       {return self.replace(args->at(1).oclAsType(String), args->at(2).oclAsType(String))}
1936     case (opName = 'restartAllStrCounter')
1937       {return self.restartAllStrCounter()}
1938     case (opName = 'rfind')
1939       {return self.rfind(args->first().oclAsType(String))}
1940     case (opName = 'startsWith')
1941       {return self.startsWith(args->first().oclAsType(String))}
1942     case (opName = 'startStrCounter')
1943       {return self.startStrCounter(args->first().oclAsType(String))}
1944     case (opName = 'substringAfter')
1945       {return self.substringAfter(args->first().oclAsType(String))}
1946     case (opName = 'substringBefore')
1947       {return self.substringBefore(args->first().oclAsType(String))}
1948     case (opName = 'quotify')
1949       {return self.quotify(args->first().oclAsType(String))}
1950     case (opName = 'trim')
1951       {return self.trim()}
1952     case (opName = 'unquotify')
1953       {return self.unquotify(args->first().oclAsType(String))}
1954     else
1955       {

1957       }
1958   };
1959   return null;
1960 }
```

```
1965 ////////////////////////////////////////////////////////////////////////////
1966 //
1967 // Integer::invoke
1968 //
1969 ////////////////////////////////////////////////////////////////////////////

1971 helper Integer::invoke(opName : String, args : Sequence(OclAny)) : OclAny {
1972   switch {
1973     case (opName = '<')
1974       {return self < args->first().oclAsType(Integer)}
1975     case (opName = '>')
1976       {return self > args->first().oclAsType(Integer)}
1977     case (opName = '=' )
1978       {return self = args->first().oclAsType(Integer)}
1979     case (opName = '<>')
1980       {return self <> args->first().oclAsType(Integer)}
1981     case (opName = '>=')
1982       {return self >= args->first().oclAsType(Integer)}
1983     case (opName = '<=')
1984       {return self <= args->first().oclAsType(Integer)}
1985     case (opName = '+')
1986       {return self + args->first().oclAsType(Integer)}
1987     case (opName = '-')
```

```
      {
        if args->size() = 0 then {
          return (-self);
        }
        else {
          return self - args->first().oclAsType(Integer);
        }
        endif;
      }
    case (opName = 'div')
      {return self.div(args->first().oclAsType(Integer))}
    case (opName = 'mod')
        {return self.mod(args->first().oclAsType(Integer))}
    case (opName = '*')
      {return self * args->first().oclAsType(Integer)}
    case (opName = '/')
      {return self / args->first().oclAsType(Integer)}
    case (opName = 'abs')
      {return self.abs()}
    case (opName = 'toString')
      {return self.toString()}
    else
      {
      }
  };
  return null;
}


////////////////////////////////////////////////////////////////////////////
//
// Boolean::invoke
//
////////////////////////////////////////////////////////////////////////////

helper Boolean::invoke(opName : String, args : Sequence(OclAny)) : OclAny {
  switch {
    case (opName = '=')
      {return self = args->first().oclAsType(Boolean)}
    case (opName = '<>')
      {return self <> args->first().oclAsType(Boolean)}
    case (opName = 'not')
      {return self.not()}
    case (opName = 'and')
      {return self and args->first().oclAsType(Boolean)}
    case (opName = 'or')
      {return self or args->first().oclAsType(Boolean)}
    case (opName = 'implies')
      {return self implies args->first().oclAsType(Boolean)}
    case (opName = 'xor')
      {return self xor args->first().oclAsType(Boolean)}
    else
      {
      }
  };
  return null;
}


////////////////////////////////////////////////////////////////////////////
```

```
//
// Real::invoke
//
////////////////////////////////////////////////////////////////////////////////

helper Real::invoke(opName : String, args : Sequence(OclAny)) : OclAny {
  switch {
    case (opName = '=')
      {return self = args->first().oclAsType(Real)}
    case (opName = '<>')
      {return self <> args->first().oclAsType(Real)}
    case (opName = '>')
      {return self > args->first().oclAsType(Real)}
    case (opName = '<')
      {return self < args->first().oclAsType(Real)}
    case (opName = '<=')
      {return self <= args->first().oclAsType(Real)}
    case (opName = '>=')
      {return self >= args->first().oclAsType(Real)}
    case (opName = '+')
      {return self + args->first().oclAsType(Real)}
    case (opName = '-')
      {return self - args->first().oclAsType(Real)}
    case (opName = '*')
      {return self * args->first().oclAsType(Real)}
    case (opName = '/')
      {return self / args->first().oclAsType(Real)}
    case (opName = 'max')
      {return self.max(args->first().oclAsType(Real))}
    case (opName = 'min')
      {return self.min(args->first().oclAsType(Real))}
    case (opName = 'abs')
      {return self.abs()}
    case (opName = 'floor')
      {return self.floor()}
    case (opName = 'round')
      {return self.round()}
    case (opName = 'toString')
      {return self.toString()}
    else
      {
      }
  };
  return null;
}


////////////////////////////////////////////////////////////////////////////////
//
// Model::invoke
//
////////////////////////////////////////////////////////////////////////////////

helper Model::invoke(opName : String, args : Sequence(OclAny)) : OclAny {
  switch {
    case (opName = 'copy')
      {return self.copy()}
    case (opName = 'createEmptyModel')
      {return self.createEmptyModel()}
    case (opName = 'objects')
      {return new CollectionWrapper(self.objects())}
```

```
2110      case (opName = 'objectsOfType')
2111        {
2112          var arg := args->first().oclAsType(EClassifier);
2113          var res := self.objects()->oclAsType(EObject)->xselect(o|o.eClass().name = arg.name)->asOrderedSet();
2114          return new CollectionWrapper(res);
2115        }
2116      case (opName = 'removeElement')
2117        { return self.removeElement(args->first().oclAsType(Element))}
2118      case (opName = 'rootObjects')
2119        {return new CollectionWrapper(self.rootObjects())}
2120      case (opName = '=')
2121        {return self = args->first()}
2122      case (opName = '<>')
2123        {return self <> args->first()}

2125    };
2126    return null;
2127  }


2130  ////////////////////////////////////////////////////////////////////////////
2131  //
2132  // EObject::invoke
2133  //
2134  ////////////////////////////////////////////////////////////////////////////

2136  helper ecore::EObject::invoke(opName : String, args : Sequence(OclAny)) : OclAny {
2137    switch {
2138      case (opName = 'oclIsKindOf')
2139        {
2140          var typeArg := args->first().oclAsType(EClass);
2141          return typeArg.name = self.eClass().name;
2142        }
2143      case (opName = 'oclAsType')
2144        {
2145          var typeArg := args->first().oclAsType(EClass);
2146          if typeArg.name = self.eClass().name then
2147            return self
2148          endif;
2149          return null;
2150        }
2151    };

2153    return null;
2154  }


2157  ////////////////////////////////////////////////////////////////////////////
2158  //
2159  // CollectionWrapper::invoke
2160  //
2161  ////////////////////////////////////////////////////////////////////////////

2163  helper CollectionWrapper::invoke(opName : String, args : Sequence(OclAny)) : OclAny {
2164    var c := self.collection();
2165    switch {
2166      case (opName = 'count') { return c->count(args->first()); }
2167      case (opName = 'excludes') {return c->excludes(args->first());}
2168      case (opName = 'excludesAll')
2169        {
2170          var a := args->first()->asSequence();
```

```
2171          return c->excludesAll(a);
2172        }
2173      case (opName = 'includes') {return c->includes(args->first());}
2174      case (opName = 'includesAll') {return c->includesAll(args->first()->asSequence());}
2175      case (opName = 'isEmpty') {return c->isEmpty();}
2176      case (opName = 'notEmpty') {return c->notEmpty();}
2177      case (opName = 'product') {return new CollectionWrapper(c->product(args->first()->asSequence()));}
2178      case (opName = 'sum') {return c->sum();}
2179      case (opName = 'size') {return c->size();}
2180      else
2181        {
2182          switch
2183          {
2184            case (self.type = 'Bag')
2185              {
2186                return bagInvoke(self.bag, opName, args);
2187              }
2188            case (self.type = 'Set')
2189              {
2190                return setInvoke(self.set, opName, args);
2191              }
2192            case (self.type = 'OrderedSet')
2193              {
2194                return ordInvoke(self.ord, opName, args);
2195              }
2196            case (self.type = 'Sequence')
2197              {
2198                return seqInvoke(self.seq, opName, args);
2199              }
2200            case (self.type = 'List')
2201              {
2202                return listInvoke(self.list, opName, args);
2203              }

2205          };
2206        }

2208    };
2209  }

2211  ////////////////////////////////////////////////////////////////////////////
2212  //
2213  // listInvoke
2214  //
2215  ////////////////////////////////////////////////////////////////////////////

2217  helper listInvoke(list : List(OclAny), opName : String, args : Sequence(OclAny)) : OclAny {
2218    switch {
2219      case (opName = 'append')
2220        {
2221          list->append(args->first());
2222          return new CollectionWrapper(list);
2223        }
2224      case (opName = 'prepend')
2225        {
2226          list->prepend(args->first());
2227          return new CollectionWrapper(list);
2228        }
2229      case (opName = 'insertAt')
2230        {
2231          list->insertAt(
```

```
            args->first(),
            args->at(2).oclAsType(Integer)
        );
        return new CollectionWrapper(list);
      }
    case (opName = 'joinfields')
      {
        list->joinfields(
          args->first().oclAsType(String),
          args->at(2).oclAsType(String),
          args->at(3).oclAsType(String)
        );
      }
    case (opName = 'xselect' or opName = 'select')
      {
        var expr := args->at(2).oclAsType(ImperativeIterateExp);
        var env := args->first().oclAsType(Environment);
        var iteratorname := expr.iterator->first().oclAsType(Variable).name;
        var col :=
          list->xselect(i_|
            expr.condition.eval(
              new Environment(iteratorname, i_, env)
            ).oclAsType(Boolean)
        );
        var resList : List(OclAny) := List{};
        col->forEach(e) {
          resList->append(e);
        };
        return new CollectionWrapper(resList);
      }
  };
  return null;
}



////////////////////////////////////////////////////////////////////////////
//
// seqInvoke
//
////////////////////////////////////////////////////////////////////////////

helper seqInvoke(seq : Sequence(OclAny), opName : String, args : Sequence(OclAny)) : OclAny {
  switch {
    case (opName = '=') {return seq = args->first()->asSequence();}
    case (opName = '<>') {return seq <> args->first()->asSequence();}
    case (opName = 'union') {return new CollectionWrapper(seq->union(args->first()->asSequence()));}
    case (opName = 'append') {return new CollectionWrapper(seq->append(args->first()));}
    case (opName = 'prepend') {return new CollectionWrapper(seq->prepend(args->first()));}
    case (opName = 'insertAt')
      {
        return new CollectionWrapper(
          seq->insertAt(
            args->first().oclAsType(Integer),
            args->at(2)
          )
        );
      }
    case (opName = 'subSequence')
      {
        return new CollectionWrapper(
```

```
2293            seq->subSequence(
2294              args->first().oclAsType(Integer),
2295              args->at(2).oclAsType(Integer)
2296            )
2297          );
2298        }
2299      case (opName = 'at') {return seq->at(args->first().oclAsType(Integer));}
2300      case (opName = 'indexOf') {return seq->indexOf(args->first());}
2301      case (opName = 'including') {return seq->includes(args->first());}
2302      case (opName = 'excluding') {return new CollectionWrapper(seq->excluding(args->first()));}
2303      case (opName = 'first')
2304        {
2305          var res := seq->first();
2306          return res;
2307        }
2308      case (opName = 'last') {return seq->last();}
2309      case (opName = 'flatten') {return new CollectionWrapper(seq->flatten());}
2310      case (opName = 'asSequence') {return new CollectionWrapper(seq);}
2311      case (opName = 'asSet') {return new CollectionWrapper(seq->asSet());}
2312      case (opName = 'asBag') {return new CollectionWrapper(seq->asBag());}
2313      case (opName = 'asOrderedSet') {return new CollectionWrapper(seq->asOrderedSet());}
2314      case (opName = 'asList') {return new CollectionWrapper(seq->asList());}
2315      case (opName = 'xselect' or opName = 'select')
2316        {
2317          var expr := args->at(2).oclAsType(ImperativeIterateExp);
2318          var env := args->first().oclAsType(Environment);
2319          var iteratorname := expr.iterator->first().oclAsType(Variable).name;
2320          return new CollectionWrapper(
2321            seq->xselect(i_|
2322              expr.condition.eval(
2323                new Environment(iteratorname, i_, env)
2324              ).oclAsType(Boolean)
2325            )
2326          );
2327        }
2328    };

2330    return null;
2331  }



2335  ////////////////////////////////////////////////////////////////////////////
2336  //
2337  // ordInvoke
2338  //
2339  ////////////////////////////////////////////////////////////////////////////

2341  helper ordInvoke(ord : OrderedSet(OclAny), opName : String, args : Sequence(OclAny)) : OclAny {
2342    switch {
2343      case (opName = '=') {return ord = args->first()->asOrderedSet();}
2344      case (opName = '<>') {return ord <> args->first()->asOrderedSet();}
2345      case (opName = 'union') {return new CollectionWrapper(ord->union(args->first()->asOrderedSet()));}
2346      case (opName = 'append') {return new CollectionWrapper(ord->append(args->first()));}
2347      case (opName = 'prepend') {return new CollectionWrapper(ord->prepend(args->first()));}
2348      case (opName = 'insertAt')
2349        {
2350          return new CollectionWrapper(
2351            ord->insertAt(
2352              args->first().oclAsType(Integer),
2353              args->at(2)
```

```
          )
        );
      }
      case (opName = 'subOrderedSet')
      {
        return new CollectionWrapper(
          ord->subOrderedSet(
            args->first().oclAsType(Integer),
            args->at(2).oclAsType(Integer)
          )
        );
      }
    case (opName = 'at') {return ord->at(args->first().oclAsType(Integer));}
    case (opName = 'indexOf') {return ord->indexOf(args->first());}
    case (opName = 'including') {return ord->includes(args->first());}
    case (opName = 'excluding') {return new CollectionWrapper(ord->excluding(args->first()));}
    case (opName = '-') {return new CollectionWrapper(ord - args->first()->asOrderedSet());}
    case (opName = 'union') {return new CollectionWrapper(ord->union(args->first()->asSet()));}
    case (opName = 'intersection') {return new CollectionWrapper(ord->intersection(args->first()->asSet()));}
    case (opName = 'symmetricDifference') {return new CollectionWrapper(ord->symmetricDifference(args->first()->asSet()))
        ;}
    case (opName = 'first') {return ord->first();}
    case (opName = 'last') {return ord->last();}
    case (opName = 'flatten') {return new CollectionWrapper(ord->flatten());}
    case (opName = 'asSequence') {return new CollectionWrapper(ord->asSequence());}
    case (opName = 'asSet') {return new CollectionWrapper(ord->asSet());}
    case (opName = 'asBag') {return new CollectionWrapper(ord->asBag());}
    case (opName = 'asOrderedSet') {return new CollectionWrapper(ord);}
    case (opName = 'asList') {return new CollectionWrapper(ord->asList());}
    case (opName = 'xselect' or opName = 'select')
      {
        var expr := args->at(2).oclAsType(ImperativeIterateExp);
        var env := args->first().oclAsType(Environment);
        var iteratorname := expr.iterator->first().oclAsType(Variable).name;
        return new CollectionWrapper(
          ord->xselect(i_|
            expr.condition.eval(
              new Environment(iteratorname, i_, env)
            ).oclAsType(Boolean)
          )
        );
      }
  };

  return null;
}


////////////////////////////////////////////////////////////////////////////
//
// setInvoke
//
////////////////////////////////////////////////////////////////////////////

helper setInvoke(set : Set(OclAny), opName : String, args : Sequence(OclAny)) : OclAny {
  var firstArg : Set(OclAny);
  if args->first().oclIsKindOf(CollectionWrapper) then {
    firstArg := args->first().asSet();
  }
  else {
    firstArg := args->first()->asSet();
```

```
2414   }
2415   endif;
2416   switch {
2417     case (opName = '=') {return set = firstArg;}
2418     case (opName = '<>') {return set <> firstArg;}
2419     case (opName = 'union') {return new CollectionWrapper(set->union(firstArg));}
2420     case (opName = 'intersection') {return new CollectionWrapper(set->intersection(firstArg));}
2421     case (opName = '-') {return new CollectionWrapper(set - firstArg);}
2422     case (opName = 'including') {return set->includes(args->first());}
2423     case (opName = 'excluding') {return new CollectionWrapper(set->excluding(args->first()));}
2424     case (opName = 'symmetricDifference') {return new CollectionWrapper(set->symmetricDifference(firstArg));}
2425     case (opName = 'flatten') {return new CollectionWrapper(set->flatten());}
2426     case (opName = 'asSequence') {return new CollectionWrapper(set->asSequence());}
2427     case (opName = 'asSet') {return new CollectionWrapper(set);}
2428     case (opName = 'asBag') {return new CollectionWrapper(set->asBag());}
2429     case (opName = 'asOrderedSet') {return new CollectionWrapper(set->asOrderedSet());}
2430     case (opName = 'asList') {return new CollectionWrapper(set->asList());}
2431     case (opName = 'xselect' or opName = 'select')
2432       {
2433         var expr := args->at(2).oclAsType(ImperativeIterateExp);
2434         var env := args->first().oclAsType(Environment);
2435         var iteratorname := expr.iterator->first().oclAsType(Variable).name;
2436         return new CollectionWrapper(
2437           set->xselect(i_|
2438             expr.condition.eval(
2439               new Environment(iteratorname, i_, env)
2440             ).oclAsType(Boolean)
2441           )
2442         );
2443       }
2444   };

2446   return null;
2447 }


2450 ////////////////////////////////////////////////////////////////////////////
2451 //
2452 // bagInvoke
2453 //
2454 ////////////////////////////////////////////////////////////////////////////

2456 helper bagInvoke(bag : Bag(OclAny), opName : String, args : Sequence(OclAny)) : OclAny {
2457   switch {
2458     case (opName = '=') {return bag = args->first()->asBag();}
2459     case (opName = '<>') {return bag <> args->first()->asBag();}
2460     case (opName = 'union')
2461       {
2462         return new CollectionWrapper(
2463           bag->union(
2464             args->first()->asBag()
2465           )
2466         );

2468       }
2469     case (opName = 'intersection') {return new CollectionWrapper(bag->intersection(args->first()->asBag()));}
2470     case (opName = 'including') {return bag->includes(args->first());}
2471     case (opName = 'excluding') {return new CollectionWrapper(bag->excluding(args->first()));}
2472     case (opName = 'flatten') {return new CollectionWrapper(bag->flatten());}
2473     case (opName = 'asSequence') {return new CollectionWrapper(bag->asSequence());}
2474     case (opName = 'asSet') {return new CollectionWrapper(bag->asSet());}
```

```
2475     case (opName = 'asBag') {return new CollectionWrapper(bag);}
2476     case (opName = 'asOrderedSet') {return new CollectionWrapper(bag->asOrderedSet());}
2477     case (opName = 'asList') {return new CollectionWrapper(bag->asList());}
2478     case (opName = 'xselect' or opName = 'select')
2479       {
2480         var expr := args->at(2).oclAsType(ImperativeIterateExp);
2481         var env := args->first().oclAsType(Environment);
2482         var iteratorname := expr.iterator->first().oclAsType(Variable).name;
2483         return new CollectionWrapper(
2484           bag->xselect(i_|
2485             expr.condition.eval(
2486               new Environment(iteratorname, i_, env)
2487             ).oclAsType(Boolean)
2488           )
2489         );
2490       }
2491   };

2493   return null;
2494 }

2496 ////////////////////////////////////////////////////////////////////////////
2497 //
2498 // OclAny::print
2499 //
2500 ////////////////////////////////////////////////////////////////////////////

2502 helper OclAny::print() : String {
2503   return self.repr();
2504 }


2507 ////////////////////////////////////////////////////////////////////////////////////////
2508 //
2509 //  lval Functions
2510 //
2511 ////////////////////////////////////////////////////////////////////////////////////////


2514 ////////////////////////////////////////////////////////////////////////////
2515 //
2516 // OCLExpression::lval
2517 //
2518 ////////////////////////////////////////////////////////////////////////////

2520 helper ocl::ecore::OCLExpression::lval(env : Environment) : OclAny {
2521   return null;
2522 }


2525 ////////////////////////////////////////////////////////////////////////////
2526 //
2527 // OCLExpression::lval
2528 //
2529 ////////////////////////////////////////////////////////////////////////////

2531 helper ocl::expressions::OCLExpression::lval(env : Environment) : OclAny {
2532   return null;
2533 }
```

```
2536 ////////////////////////////////////////////////////////////////////////////
2537 //
2538 // VariableExp::lval
2539 //
2540 ////////////////////////////////////////////////////////////////////////////

2542 helper VariableExp::lval(env : Environment) : OclAny {
2543   return self.name;
2544 }


2547 ////////////////////////////////////////////////////////////////////////////
2548 //
2549 // PropertyCallExp::lval
2550 //
2551 ////////////////////////////////////////////////////////////////////////////

2553 helper PropertyCallExp::lval(env : Environment) : OclAny {
2554   return self.source.lval(env);
2555 }


2558 ////////////////////////////////////////////////////////////////////////////////////////
2559 //
2560 //  eval Functions
2561 //
2562 ////////////////////////////////////////////////////////////////////////////////////////


2565 ////////////////////////////////////////////////////////////////////////////
2566 //
2567 // OperationalTransformation::eval
2568 //
2569 ////////////////////////////////////////////////////////////////////////////

2571 helper OperationalTransformation::eval(env : Environment) : OclAny {
2572   return self.entry.eval(env);
2573 }


2576 ////////////////////////////////////////////////////////////////////////////
2577 //
2578 // EntryOperation::eval
2579 //
2580 ////////////////////////////////////////////////////////////////////////////

2582 helper EntryOperation::eval(env : Environment) : OclAny {
2583   return self.body.eval(env);
2584 }


2587 ////////////////////////////////////////////////////////////////////////////
2588 //
2589 // OperationBody::eval
2590 //
2591 ////////////////////////////////////////////////////////////////////////////

2593 helper OperationBody::eval(env : Environment) : OclAny {
2594   var lastVal : OclAny;
2595   var retVal : OclAny;
2596   var retEnv := new Environment('$__return', null, env);
```

```
2597   self.content->forEach(statement) {
2598     lastVal := statement.eval(retEnv);
2599     retVal := retEnv.get('$__return').value();
2600     if (retVal <> null) then {
2601       return retVal;
2602     }
2603     endif;
2604   };
2605   return lastVal;
2606 }


2609 ////////////////////////////////////////////////////////////////////////////
2610 //
2611 // OCLExpression::eval
2612 //
2613 ////////////////////////////////////////////////////////////////////////////

2615 helper ocl::expressions::OCLExpression::eval(env : Environment): OclAny {
2616   return null;
2617 }


2620 ////////////////////////////////////////////////////////////////////////////
2621 //
2622 // ReturnExp::eval
2623 //
2624 ////////////////////////////////////////////////////////////////////////////

2626 helper ReturnExp::eval(env : Environment) : OclAny {
2627   var retVal := self.value.eval(env);
2628   env.put('$__return', createFrame(retVal));
2629   return retVal;
2630 }


2633 ////////////////////////////////////////////////////////////////////////////
2634 //
2635 // OclAny::toEObject
2636 //
2637 ////////////////////////////////////////////////////////////////////////////

2639 helper OclAny::toEObject() : EObject {
2640   return null;
2641 }


2644 ////////////////////////////////////////////////////////////////////////////
2645 //
2646 // String::toEObject
2647 //
2648 ////////////////////////////////////////////////////////////////////////////

2650 helper String::toEObject() : EObject {
2651   return self.oclAsType(EObject);
2652 }


2655 ////////////////////////////////////////////////////////////////////////////
2656 //
2657 // Real::toEObject
```

```
2658 //
2659 ////////////////////////////////////////////////////////////////////////////////

2661 helper Real::toEObject() : EObject {
2662   return self.oclAsType(EObject);
2663 }


2666 ////////////////////////////////////////////////////////////////////////////////
2667 //
2668 // Integer::toEObject
2669 //
2670 ////////////////////////////////////////////////////////////////////////////////

2672 helper Integer::toEObject() : EObject {
2673   return self.oclAsType(EObject);
2674 }


2677 ////////////////////////////////////////////////////////////////////////////////
2678 //
2679 // Boolean::toEObject
2680 //
2681 ////////////////////////////////////////////////////////////////////////////////

2683 helper Boolean::toEObject() : EObject {
2684   return self.oclAsType(EObject);
2685 }


2688 ////////////////////////////////////////////////////////////////////////////////
2689 //
2690 // EObject::toEObject
2691 //
2692 ////////////////////////////////////////////////////////////////////////////////

2694 helper EObject::toEobject() : EObject {
2695   return self;
2696 }



2700 ////////////////////////////////////////////////////////////////////////////////
2701 //
2702 // OperationCallExp::eval
2703 //
2704 ////////////////////////////////////////////////////////////////////////////////

2706 helper ocl::ecore::OperationCallExp::eval(env : Environment) : OclAny {
2707   var src := self.source.eval(env);

2709   if self.referredOperation.oclIsKindOf(Helper) then {
2710     var helpOp := self.referredOperation.oclAsType(Helper);
2711     var context : Environment;
2712     if src.oclIsInvalid() then
2713       context := new Environment(env)
2714     else
2715       context := new Environment('self', src, env)
2716     endif;
2717     var i := 1;
2718     while (i <= self.argument->size()) {
```

```
2719        var arg := self.argument->at(i).eval(env);
2720        var argName := helpOp.eParameters->at(i).name;
2721        context.put(argName, createFrame(arg));
2722        i := i + 1;
2723      };

2725      return helpOp.body.eval(context);
2726    }
2727    endif;
2728    var args := self.argument->eval(env);
2729    var res := src.invoke(self.referredOperation.oclAsType(EOperation), args);
2730    return res;
2731  }


2734  ////////////////////////////////////////////////////////////////////////////
2735  //
2736  // MappingBody::eval
2737  //
2738  ////////////////////////////////////////////////////////////////////////////

2740  helper MappingBody::eval(env : Environment) : OclAny {
2741    var initPart := self.initSection.eval(env);
2742    var contentPart := self.content->eval(env);
2743    var endPart := self.endSection.eval(env);
2744    var variablePart := self.variable;
2745    return contentPart->first();
2746  }


2749  ////////////////////////////////////////////////////////////////////////////
2750  //
2751  // MappingParameter::eval
2752  //
2753  ////////////////////////////////////////////////////////////////////////////

2755  helper MappingParameter::eval(env : Environment) : OclAny {
2756    var init_ := self.initExpression.eval(env);
2757    return null;
2758  }


2761  ////////////////////////////////////////////////////////////////////////////
2762  //
2763  // MappingOperation::eval
2764  //
2765  ////////////////////////////////////////////////////////////////////////////

2767  helper MappingOperation::eval(env : Environment) : OclAny {
2768    var whenPart := self._when.eval(env)->last().oclAsType(Boolean);
2769    if  whenPart = false then {
2770      return null;
2771    }
2772    endif;
2773    var resultVar := self.result.name;
2774    var resultType := self.result.eType;
2775    var resFrame := new ObjectFrame(null);
2776    var resEnv := new Environment(env);
2777    resEnv.localEnv->put('result', resFrame);

2779    var contextPart := self.context.eval(env);
```

```
2781   var bodyPart := self.body.eval(resEnv);
2782   //var srcType := srcObject.eClass();
2783   //var feature := srcType.getEStructuralFeature(self.referredProperty.oclAsType(EStructuralFeature).name);
2784   //return srcObject.eGet(feature);

2786   var res := resEnv.get('result').value().oclAsType(EObject);
2787   var wherePart := self._where.eval(env);
2788   return res;
2789 }


2792 ////////////////////////////////////////////////////////////////////////////
2793 //
2794 // MappingCallExp::eval
2795 //
2796 ////////////////////////////////////////////////////////////////////////////

2798 helper MappingCallExp::eval(env : Environment) : OclAny {
2799   mappingCallLevel := mappingCallLevel + 1;
2800   var src := self.source.eval(env);
2801   var type := self.eType.name;
2802   var op := self.referredOperation.oclAsType(MappingOperation);
2803   var args := self.argument->eval(env).oclAsType(CollectionWrapper).collection();
2804   var newEnv := new Environment('self', src, env);

2806   var i := 1;
2807   while (i <= self.argument->size()) {
2808     var arg := self.argument->at(i).eval(env);
2809     var argName := op.eParameters->at(i).name;
2810     newEnv.put(argName, createFrame(arg));
2811     i := i + 1;
2812   };

2814   var res := op.eval(newEnv);
2815   mappingCallLevel := mappingCallLevel - 1;
2816   if mappingCallLevel = 0 and res.oclIsKindOf(EObject) then {
2817     var resObj := res.oclAsType(EObject);
2818     resObj.map writeObject();
2819   }
2820   endif;
2821   return res;
2822 }


2825 ////////////////////////////////////////////////////////////////////////////
2826 //
2827 // BlockExp::eval
2828 //
2829 ////////////////////////////////////////////////////////////////////////////

2831 helper BlockExp::eval(env : Environment) :OclAny {
2832   self.body->eval(new Environment(env));
2833   return null;
2834 }


2837 ////////////////////////////////////////////////////////////////////////////
2838 //
2839 // LogExp::eval
2840 //
```

```
2841 ////////////////////////////////////////////////////////////////////////////

2843 helper LogExp::eval(env : Environment) : OclAny {
2844   if (self.condition = null or self.condition.eval(env).oclAsType(Boolean) <> false) then {
2845     var output : String := "";
2846     self.argument->forEach(a) {
2847       var res := a.eval(env);
2848       output := output + a.eval(env).print();
2849     };
2850     log(output);
2851   }
2852   endif;
2853   return null;
2854 }


2857 ////////////////////////////////////////////////////////////////////////////
2858 //
2859 // AssignExp::eval
2860 //
2861 ////////////////////////////////////////////////////////////////////////////

2863 helper AssignExp::eval(env : Environment) : OclAny {

2865   var lValue := self.left.lval(env).oclAsType(String);
2866   var rValue := self.value->first().eval(env);
2867   if (self.left.oclIsKindOf(PropertyCallExp)) then {
2868     var propExpr := self.left.oclAsType(PropertyCallExp);
2869     var srcFrameName := propExpr.source.getName();
2870     var lFrame := env.get(lValue).oclAsType(ObjectFrame);
2871     var feature := propExpr.referredProperty.oclAsType(EStructuralFeature);
2872     var obj := lFrame.value().oclAsType(EObject);
2873     if rValue.oclIsKindOf(CollectionWrapper) and
2874       (feature.upperBound > 1 or feature.upperBound = -1 or
2875         feature.eType.oclIsKindOf(CollectionType)) then {
2876       var col := rValue.oclAsType(CollectionWrapper);
2877       switch {
2878         case (col.type = 'Sequence')
2879         {setMultiFeature(obj, feature.name, col.seq);}
2880         case (col.type = 'OrderedSet')
2881         {setMultiFeature(obj, feature.name, col.ord);}
2882         case (col.type = 'Bag')
2883         {setMultiFeature(obj, feature.name, col.bag);}
2884         case (col.type = 'Set')
2885         {setMultiFeature(obj, feature.name, col.set);}
2886         case (col.type = 'List')
2887         {setMultiFeature(obj, feature.name, col.list);}
2888       };

2890     }
2891     else {
2892       obj.eSet(feature, rValue);
2893     }
2894     endif;
2895     lFrame.objValue := obj;
2896     env.put(srcFrameName, lFrame);
2897   }
2898   else
2899     if (self.left.oclIsKindOf(VariableExp)) then {
2900       var srcFrameName := self.left.getName();
2901       env.put(srcFrameName, createFrame(rValue));
```

```
    }
    endif
  endif;
  //env.put(lValue.)
  return rValue;
}


////////////////////////////////////////////////////////////////////////////
//
// IteratorExp::eval
//
////////////////////////////////////////////////////////////////////////////

helper IteratorExp::eval(env : Environment) : OclAny {
  var sourceCol_ := self.source.eval(env).oclAsType(CollectionWrapper);
  var source_ := sourceCol_.collection();
  var type_ := sourceCol_.type;
  var iterator_ := self.iterator;
  var itername := self.iterator->first().oclAsType(Variable).name;
  var res : OclAny;

  switch {
    case (self.name = 'select') {
      //var condition_ : Function := self.makeCondition(env);
      //var arg1 := self.condition.oclAsType(OperationCallExp).argument->first().eval(env);
      return sourceCol_.invoke(self.name, Sequence{env, self});
/*
      if type_ = 'List' then {
        var reswrap_ := new CollectionWrapper();
        var resList_ : List(OclAny) := List{};
        var resCol_ : Collection(OclAny);
        if self.condition.oclIsKindOf(TypeExp) then {
                  resCol_ := sourceCol_.list->
                xselect(i_|
                  self.condition.eval(
                    new Environment(itername, i_, env)
                  ).oclAsType(Boolean)
                );
        }
        endif;
        resCol_ := sourceCol_.list->
                xselect(i_|
                  self.condition.eval(
                    new Environment(itername, i_, env)
                  ).oclAsType(Boolean)
                );
        resCol_->forEach(e) {
          resList_->append(e);
        };

        reswrap_.type := 'List';
        reswrap_.list := resList_;
        res := reswrap_;
      }
      else {
        res := new CollectionWrapper (
            source_->xselect(i_|
              self.condition.eval(
                new Environment(itername, i_, env)
              ).oclAsType(Boolean)
```

```
            )
        );
      }
      endif;
*/
    }
    case (self.name = 'collect') {
      res := new CollectionWrapper(
            source_->xcollect(i_|
                  self.body.eval(
                    new Environment(itername, i_, env)
                  )
            )
      );

    }
  };
  return res;
}



////////////////////////////////////////////////////////////////////////////
//
// ImperativeIterateExp::eval
//
////////////////////////////////////////////////////////////////////////////

helper ImperativeIterateExp::eval(env : Environment) : OclAny {
  var sourceCol_ := self.source.eval(env).oclAsType(CollectionWrapper);
  var source_ := sourceCol_.collection();
  var type_ := sourceCol_.type;
  var iterator_ := self.iterator;
  var itername := self.iterator->first().oclAsType(Variable).name;
  var res : OclAny;

  switch {
    case (self.name = 'xselect') {
      //var condition_ : Function := self.makeCondition(env);
      //var arg1 := self.condition.oclAsType(OperationCallExp).argument->first().eval(env);
      return sourceCol_.invoke(self.name, Sequence{env, self});
/*
      if type_ = 'List' then {
        var reswrap_ := new CollectionWrapper();
        var resList_ : List(OclAny) := List{};
        var resCol_ : Collection(OclAny);
        if self.condition.oclIsKindOf(TypeExp) then {
                  resCol_ := sourceCol_.list->
                xselect(i_|
                  self.condition.eval(
                    new Environment(itername, i_, env)
                  ).oclAsType(Boolean)
                );
        }
        endif;
        resCol_ := sourceCol_.list->
                xselect(i_|
                  self.condition.eval(
                    new Environment(itername, i_, env)
                  ).oclAsType(Boolean)
                );
```

```
3024          resCol_->forEach(e) {
3025            resList_->append(e);
3026          };

3028          reswrap_.type := 'List';
3029          reswrap_.list := resList_;
3030          res := reswrap_;
3031        }
3032        else {
3033          res := new CollectionWrapper (
3034              source_->xselect(i_|
3035                self.condition.eval(
3036                  new Environment(itername, i_, env)
3037                ).oclAsType(Boolean)
3038              )
3039          );
3040        }
3041        endif;
3042  */
3043      }
3044      case (self.name = 'xcollect') {
3045        if (self.body <> null) then {
3046          res := new CollectionWrapper(
3047                source_->xcollect(i_|
3048                      self.body.eval(
3049                        new Environment(itername, i_, env)
3050                      )
3051                )
3052          );
3053        }
3054        else { // if the body is null (e.g., collection type casts)
3055          res := new CollectionWrapper(source_->xcollect(i_|i_));
3056        }
3057        endif;
3058      }
3059      case (self.name = 'selectOne') {
3060        res := source_->selectOne(i_|
3061                  self.condition.eval(
3062                    new Environment(itername, i_, env)
3063                  ).oclAsType(Boolean)
3064        );
3065      }
3066      case (self.name = 'collectOne') {
3067        res := source_->collectOne(i_|
3068                  self.body.eval(
3069                    new Environment(itername, i_, env)
3070                  ).oclAsType(Boolean)
3071        );
3072      }
3073      case (self.name = 'collectselect') {
3074        var target := self.target.oclAsType(Variable).name;
3075        var iterEnv := new Environment(env);
3076        res := new CollectionWrapper(
3077                source_->collectselect(
3078                  i_;
3079                  t = self.body.eval(
3080                      iterEnv.put(itername, createFrame(i_))
3081                    )
3082                  | self.condition.eval(
3083                      iterEnv.put(target, createFrame(t))
3084                    ).oclAsType(Boolean)
```

```
3085                )
3086       );
3087     }
3088     case (self.name = 'collectselecOne') {
3089       var target := self.target.oclAsType(Variable).name;
3090       var iterEnv := new Environment(env);
3091       res := source_->collectselectOne(
3092                 i_;
3093                 t = self.body.eval(
3094                     iterEnv.put(itername, createFrame(i_))
3095                   )
3096                 | self.condition.eval(
3097                     iterEnv.put(target, createFrame(t))
3098                   ).oclAsType(Boolean)
3099       );

3101     }
3102   };
3103   return res;
3104 }


3107 ////////////////////////////////////////////////////////////////////////////
3108 //
3109 // ForExp::eval
3110 //
3111 ////////////////////////////////////////////////////////////////////////////

3113 helper ForExp::eval(env : Environment) : OclAny {
3114   var source_ := self.source.eval(env).oclAsType(CollectionWrapper).collection();
3115   var itername := self.iterator->first().oclAsType(Variable).name;
3116   var iterEnv := new Environment(env);
3117   switch {
3118     case (self.name = 'forEach') {
3119       if (self.condition = null) then {
3120         source_->forEach(i_) {
3121           self.body.eval(iterEnv.put(itername,createFrame(i_)));
3122         };
3123       }
3124       else {
3125         source_->forEach(i_|
3126                   self.condition.eval(
3127                     iterEnv.put(
3128                       itername, createFrame(i_)
3129                     )
3130                   ).oclAsType(Boolean)
3131             ) {
3132           self.body.eval(iterEnv);
3133         };
3134       }
3135       endif;
3136     }
3137     case (self.name = 'forOne') {
3138       if self.condition = null then {
3139         source_->forOne(i_) {
3140           self.body.eval(iterEnv.put(itername, createFrame(i_)));
3141         }
3142       }
3143       else {
3144         source_->forOne(i_|
3145                   self.condition.eval(
```

```
                    iterEnv.put(
                      itername, createFrame(i_)
                    )
                  ).oclAsType(Boolean)
            ) {
          self.body.eval(iterEnv);
        };
      }
      endif;
    }
  };
  return null;
}


////////////////////////////////////////////////////////////////////////////////
//
// OclAny::makeExp
//
////////////////////////////////////////////////////////////////////////////////

helper OclAny::makeExp() : OCLExpression {
  if self.oclIsKindOf(EObject) then {
    var obj := self.oclAsType(EObject);
    var x := obj.makeExp();
    return x;
  }
  endif;
  return null;
}


////////////////////////////////////////////////////////////////////////////////
//
// OCLExpression::makeExp
//
////////////////////////////////////////////////////////////////////////////////

helper OCLExpression::makeExp() : OCLExpression {
  return self;
}


////////////////////////////////////////////////////////////////////////////////
//
// EObject::makeExp
//
////////////////////////////////////////////////////////////////////////////////

helper EObject::makeExp() : OCLExpression {
  var path := self.path();
  var objInModel := inputModel.getObject(path);
  var obj := self;
  if objInModel <> null and not objInModel.oclIsInvalid() then {
    obj := path.oclAsType(EObject);
  }
  endif;
  var literalExp : ObjectExp := new ObjectExp();
  literalExp.body := new ConstructorBody();
  obj.eClass().eAllAttributes->forEach(attr) {
    var attrValue : OclAny;
```

```
3207      if (attr.eType.oclIsKindOf(CollectionType)) then {
3208        var col := Sequence{};
3209        getMultiFeature(obj, attr.name, col);
3210        attrValue := new CollectionWrapper(col);
3211      }
3212      else {
3213        attrValue := obj.eGet(attr);
3214      }
3215      endif;
3216      literalExp.body.content +=
3217        object AssignExp {
3218          left := object VariableExp {
3219            referredVariable := object Variable {
3220              name := attr.name;
3221            };
3222            name := referredVariable.oclAsType(Variable).name;
3223          };
3224          value := attrValue.makeExp();
3225      };
3226      literalExp.instantiatedClass := obj.eClass();
3227    };

3229    return literalExp;
3230  }


3233  ////////////////////////////////////////////////////////////////////////////
3234  //
3235  // Integer::makeExp
3236  //
3237  ////////////////////////////////////////////////////////////////////////////

3239  helper Integer::makeExp() : OCLExpression {
3240    return object IntegerLiteralExp {
3241      integerSymbol := self;
3242    }
3243  }


3246  ////////////////////////////////////////////////////////////////////////////
3247  //
3248  // Real::makeExp
3249  //
3250  ////////////////////////////////////////////////////////////////////////////

3252  helper Real::makeExp() : OCLExpression {
3253    return object RealLiteralExp {
3254      realSymbol := self;
3255    }
3256  }


3259  ////////////////////////////////////////////////////////////////////////////
3260  //
3261  // String::makeExp‘
3262  //
3263  ////////////////////////////////////////////////////////////////////////////

3265  helper String::makeExp() : OCLExpression {
3266    return object StringLiteralExp {
3267      stringSymbol := self;
```

```
3268   }
3269 }


3272 ////////////////////////////////////////////////////////////////////////////
3273 //
3274 // Boolean::makeExp
3275 //
3276 ////////////////////////////////////////////////////////////////////////////

3278 helper Boolean::makeExp() : OCLExpression {
3279   return object BooleanLiteralExp {
3280     booleanSymbol := self;
3281   }
3282 }



3286 ////////////////////////////////////////////////////////////////////////////
3287 //
3288 // CollectionWrapper::makeExp
3289 //
3290 ////////////////////////////////////////////////////////////////////////////

3292 helper CollectionWrapper::makeExp() : OCLExpression {
3293   var literalExp := new CollectionLiteralExp();

3295   self.collection()->forEach(element) {
3296     literalExp.part +=
3297       object CollectionItem {
3298         item := element.makeExp();
3299       };
3300   };
3301   switch {
3302     case (self.type = 'Set')
3303       {
3304         literalExp.kind := ocl::expressions::CollectionKind::Set;
3305         literalExp.eType := object SetType {

3307         };
3308       }
3309     case (self.type = 'Sequence')
3310       {
3311         literalExp.kind := ocl::expressions::CollectionKind::Sequence;
3312         literalExp.eType := object SequenceType {

3314         };
3315       }
3316     case (self.type = 'OrderedSet')
3317       {
3318         literalExp.kind := ocl::expressions::CollectionKind::OrderedSet;
3319         literalExp.eType := object OrderedSetType {

3321         };
3322       }
3323     case (self.type = 'List')
3324       {
3325         literalExp.kind := ocl::expressions::CollectionKind::Sequence;
3326         literalExp.eType := object ListType {

3328         };
```

```
3329        }
3330      case (self.type = 'Bag')
3331        {
3332          literalExp.kind := ocl::expressions::CollectionKind::Bag;
3333          literalExp.eType := object BagType {

3335          };
3336        }
3337    };
3338    return literalExp;
3339  }


3342  ////////////////////////////////////////////////////////////////////////////
3343  //
3344  // CollectionLiteralExp::eval
3345  //
3346  ////////////////////////////////////////////////////////////////////////////

3348  helper CollectionLiteralExp::eval(env : Environment) : OclAny {
3349    var type := self.eType.name;
3350    var seq : Sequence(OclAny);
3351    self.part->forEach(p) {
3352      seq := seq->append(p.oclAsType(CollectionItem).eval(env));
3353    };
3354    var res : CollectionWrapper;

3356    switch {
3357      case (type.startsWith('Set'))
3358        {res := new CollectionWrapper(seq->asSet())}
3359      case (type.startsWith('Sequence'))
3360        {res := new CollectionWrapper(seq)}
3361      case (type.startsWith('OrderedSet'))
3362        {res := new CollectionWrapper(seq->asOrderedSet())}
3363      case (type.startsWith('Bag'))
3364        {res := new CollectionWrapper(seq->asBag())}
3365      case (type.startsWith('List'))
3366        {res := new CollectionWrapper(seq->asList())}
3367    };
3368    return res;
3369  }


3372  ////////////////////////////////////////////////////////////////////////////
3373  //
3374  // StringLiteralExp::eval
3375  //
3376  ////////////////////////////////////////////////////////////////////////////

3378  helper StringLiteralExp::eval(env : Environment) : OclAny {
3379    return self.stringSymbol;
3380  }


3383  ////////////////////////////////////////////////////////////////////////////
3384  //
3385  // IntegerLiteralExp::eval
3386  //
3387  ////////////////////////////////////////////////////////////////////////////

3389  helper IntegerLiteralExp::eval(env : Environment) : OclAny {
```

```
3390   return self.integerSymbol;
3391 }


3394 ////////////////////////////////////////////////////////////////////////////////
3395 //
3396 // BooleanLiteralExp::eval
3397 //
3398 ////////////////////////////////////////////////////////////////////////////////

3400 helper BooleanLiteralExp::eval(env : Environment) : OclAny {
3401   return self.booleanSymbol;
3402 }


3405 ////////////////////////////////////////////////////////////////////////////////
3406 //
3407 // RealLiteralExp::eval
3408 //
3409 ////////////////////////////////////////////////////////////////////////////////

3411 helper RealLiteralExp::eval(env : Environment) : OclAny {
3412   return self.realSymbol;
3413 }


3416 ////////////////////////////////////////////////////////////////////////////////
3417 //
3418 // NullLiteralExp::eval
3419 //
3420 ////////////////////////////////////////////////////////////////////////////////

3422 helper NullLiteralExp::eval(env : Environment) : OclAny {
3423   return null;
3424 }


3427 ////////////////////////////////////////////////////////////////////////////////
3428 //
3429 // CollectionItem::eval
3430 //
3431 ////////////////////////////////////////////////////////////////////////////////

3433 helper CollectionItem::eval(env: Environment) : OclAny {
3434   return self.item.eval(env);
3435 }



3439 ////////////////////////////////////////////////////////////////////////////////
3440 //
3441 // PropertyCallExp::eval
3442 //
3443 ////////////////////////////////////////////////////////////////////////////////

3445 helper PropertyCallExp::eval(env : Environment) : OclAny {
3446   var srcObject := self.source.eval(env).oclAsType(EObject);
3447   var srcType := srcObject.eClass();
3448   var feature := srcType.getEStructuralFeature(self.referredProperty.oclAsType(EStructuralFeature).name);
3449   if (feature.upperBound = -1) or (feature.upperBound > 1) or feature.eType.oclIsKindOf(CollectionType) then {
3450     var subobj := srcObject.allSubobjects();
```

```
3451     //var val := subobj->xselect(obj : EObject|obj.eContainingFeature() = feature);
3452     var val := Sequence{};
3453     getMultiFeature(srcObject, feature.name, val);
3454     var value := new CollectionWrapper(val->asOrderedSet());
3455 //    switch {
3456 //      case (feature.eType.oclIsKindOf(SetType)) value := new CollectionWrapper(val->asSet());
3457 //      case (feature.eType.oclIsKindOf(BagType)) value := new CollectionWrapper(val->asBag());
3458 //      case (feature.eType.oclIsKindOf(OrderedSetType)) value := new CollectionWrapper(val->asOrderedSet());
3459 //      case (feature.eType.oclIsKindOf(SequenceType)) value := new CollectionWrapper(val->asSequence());
3460 //      case (feature.eType.oclIsKindOf(ListType)) value := new CollectionWrapper(val->asList());
3461 //    };
3462     return value;
3463   }
3464   endif;
3465   var value := srcObject.eGet(feature);
3466   return value;
3467 }
3468
3469
3470
3471
3472 ////////////////////////////////////////////////////////////////////////////////
3473 //
3474 // VariableExp::eval
3475 //
3476 ////////////////////////////////////////////////////////////////////////////////
3477
3478 helper VariableExp::eval(env : Environment) : OclAny {
3479   return env.get(self.name).value();
3480 }
3481
3482
3483
3484 ////////////////////////////////////////////////////////////////////////////////
3485 //
3486 // VariableInitExp::eval
3487 //
3488 ////////////////////////////////////////////////////////////////////////////////
3489
3490 helper VariableInitExp::eval(env : Environment) : OclAny {
3491   var initValue := self.referredVariable.initExpression.eval(env);
3492   var frame := createFrame(initValue);
3493   env.localEnv->put(self.referredVariable.name, frame);
3494   return initValue;
3495 }
3496
3497
3498 ////////////////////////////////////////////////////////////////////////////////
3499 //
3500 // WhileExp::eval
3501 //
3502 ////////////////////////////////////////////////////////////////////////////////
3503
3504 helper WhileExp::eval(env : Environment) : OclAny {
3505   var res : OclAny := null;
3506   while(self.condition.eval(env).oclAsType(Boolean)) {
3507     res := self.body.eval(env);
3508   };
3509   return null;
3510 }
```

```
3513 ////////////////////////////////////////////////////////////////////////////
3514 //
3515 // ComputeExp::eval
3516 //
3517 ////////////////////////////////////////////////////////////////////////////

3519 helper ComputeExp::eval(env : Environment) : OclAny {
3520   var initValue := self.returnedElement.initExpression.eval(env);
3521   var context := new Environment(self.returnedElement.name, initValue, env);
3522   var body := self.body.eval(context);
3523   var ret := context.get(self.returnedElement.name);
3524   return ret.value();
3525 }


3528 ////////////////////////////////////////////////////////////////////////////
3529 //
3530 // IfExp::eval
3531 //
3532 ////////////////////////////////////////////////////////////////////////////

3534 helper IfExp::eval(env : Environment) : OclAny {
3535   return
3536     if (self.condition.eval(env).oclAsType(Boolean)) then
3537        self.thenExpression.eval(env)
3538     else
3539        self.elseExpression.eval(env)

3541     endif;
3542 }


3545 ////////////////////////////////////////////////////////////////////////////
3546 //
3547 // InstantiationExp::eval
3548 //
3549 ////////////////////////////////////////////////////////////////////////////

3551 helper InstantiationExp::eval(env : Environment) : OclAny {
3552   var type := self.instantiatedClass;
3553   var instance := type.ePackage.eFactoryInstance.create(type);
3554   return instance;
3555 }


3558 mapping EObject::writeObject() : EObject {
3559   init{
3560     var cont := self.eContainer();
3561     result := self.deepclone().oclAsType(EObject);
3562   }
3563 }


3566 ////////////////////////////////////////////////////////////////////////////
3567 //
3568 // ObjectExp::eval
3569 //
3570 ////////////////////////////////////////////////////////////////////////////

3572 helper ObjectExp::eval(env : Environment) : OclAny {
```

```
3573   var type := self.instantiatedClass;
3574   var instance := type.ePackage.eFactoryInstance.create(type);
3575   var newEnv := new Environment(self.referredObject.name, instance, env);
3576   self.body.eval(newEnv);
3577   instance := newEnv.get(self.referredObject.name).value().oclAsType(EObject);
3578   var extFrame := newEnv.get(self.extent.name);
3579   if extFrame <> null then {
3580     var extent := extFrame.value().oclAsType(Model);
3581     if self.referredObject.oclIsKindOf(MappingParameter) then {
3582       var refObj := self.referredObject.oclAsType(MappingParameter);
3583       if refObj.name = 'result' then {
3584         env.put('result', createFrame(instance));
3585         return instance;
3586       }
3587       endif;
3588     }
3589     endif;
3590     instance.map writeObject();
3591   }
3592   endif;
3593   return instance;
3594 }


3597 ////////////////////////////////////////////////////////////////////////////
3598 //
3599 // TypeExp::eval
3600 //
3601 ////////////////////////////////////////////////////////////////////////////

3603 helper TypeExp::eval(env : Environment) : OclAny {
3604   var type := self.referredType;
3605   var javatype := self.getType();
3606   return type;
3607 }


3610 ////////////////////////////////////////////////////////////////////////////
3611 //
3612 // ConstructorBody::eval
3613 //
3614 ////////////////////////////////////////////////////////////////////////////

3616 helper ConstructorBody::eval(env : Environment) : OclAny {
3617   self.content->eval(env);
3618   //self.operation;
3619   //self.variable;
3620   return null;
3621 }




3626 ////////////////////////////////////////////////////////////////////////////
3627 //
3628 // ASTNode::eval
3629 //
3630 ////////////////////////////////////////////////////////////////////////////

3632 helper ASTNode::eval(env : Environment) : OclAny {
3633   return null;
```

```
3634 }
3635 ////////////////////////////// QvtPrettyPrinter ////////////////////////////////////////

3637 property _tab_ : String = '    ';



3641 ////////////////////////////////////////////////////////////////////////////////
3642 //
3643 // printTabs
3644 //
3645 ////////////////////////////////////////////////////////////////////////////////

3647 helper printTabs(tabs : Integer) : String {
3648   var i := 0;
3649   var code := '';
3650   while (i < tabs) {
3651     code := code + _tab_;
3652     i := i + 1;
3653   };
3654   return code;
3655 }


3658 ////////////////////////////////////////////////////////////////////////////////
3659 //
3660 // printArgs
3661 //
3662 ////////////////////////////////////////////////////////////////////////////////

3664 helper printArgs(list : OrderedSet(ecore::EObject)) : String {
3665   if (list->isEmpty()) then
3666     return ""
3667   endif;
3668   var code : String := list->first().print(0);
3669   var rest := list->subOrderedSet(2, list->size());
3670   rest->forEach(expr) {
3671     code := code + ', ' + expr.print(0);
3672   };
3673   return code;
3674 }


3677 ////////////////////////////////////////////////////////////////////////////////
3678 //
3679 // printArgs
3680 //
3681 ////////////////////////////////////////////////////////////////////////////////

3683 helper printArgs(list : OrderedSet(ecore::EModelElement)) : String {
3684   if (list->isEmpty()) then
3685     return ""
3686   endif;
3687   var code : String := list->first().print(0);
3688   var rest := list->subOrderedSet(2, list->size());
3689   rest->forEach(expr) {
3690     code := code + ', ' + expr.print(0);
3691   };
3692   return code;
3693 }
```

```
3696 ////////////////////////////////////////////////////////////////////////////////
3697 //
3698 // printArgs
3699 //
3700 ////////////////////////////////////////////////////////////////////////////////

3702 helper printArgs(list : OrderedSet(ocl::expressions::CollectionLiteralPart)) : String {
3703   if (list->isEmpty()) then
3704     return ""
3705   endif;
3706   var code : String := list->first().print(0);
3707   var rest := list->subOrderedSet(2, list->size());
3708   rest->forEach(expr) {
3709     code := code + ', ' + expr.print(0);
3710   };
3711   return code;
3712 }


3715 ////////////////////////////////////////////////////////////////////////////////
3716 //
3717 // printArgs
3718 //
3719 ////////////////////////////////////////////////////////////////////////////////

3721 helper printArgs(list : OrderedSet(ocl::utilities::ASTNode)) : String {
3722   if (list->isEmpty()) then
3723     return ""
3724   endif;
3725   var code : String := list->first().print(0);
3726   var rest := list->subOrderedSet(2, list->size());
3727   rest->forEach(expr) {
3728     code := code + ', ' + expr.print(0);
3729   };
3730   return code;
3731 }


3734 ////////////////////////////////////////////////////////////////////////////////
3735 //
3736 // printExpressions
3737 //
3738 ////////////////////////////////////////////////////////////////////////////////

3740 helper printExpressions(exprList : OrderedSet(ocl::ecore::OCLExpression), tabs : Integer) : String {
3741   var code : String;
3742   exprList->forEach(expr) {
3743     code := code + printTabs(tabs) + expr.print(tabs) + ';\n';
3744   };
3745   return code;
3746 }



3750 ////////////////////////////////////////////////////////////////////////////////
3751 //
3752 // EObject::print
3753 //
3754 ////////////////////////////////////////////////////////////////////////////////
```

```
3756 helper ecore::EObject::print(tabs : Integer) : String {
3757   var nameFeature := self.eClass().getEStructuralFeature('name');
3758   return self.eGet(nameFeature).oclAsType(String);
3759 }


3762 ////////////////////////////////////////////////////////////////////////////
3763 //
3764 // EstructuralFeature::print
3765 //
3766 ////////////////////////////////////////////////////////////////////////////

3768 helper ecore::EStructuralFeature::print(tabs : Integer) : String {
3769   var code : String := self.name;
3770   return code;
3771 }


3774 ////////////////////////////////////////////////////////////////////////////
3775 //
3776 // EModelElement::print
3777 //
3778 ////////////////////////////////////////////////////////////////////////////

3780 helper ecore::EModelElement::print(tabs : Integer) : String {
3781   assert(true);
3782   return "";
3783 }


3786 ////////////////////////////////////////////////////////////////////////////
3787 //
3788 // ASTNode::print
3789 //
3790 ////////////////////////////////////////////////////////////////////////////

3792 helper ocl::utilities::ASTNode::print(tabs : Integer) : String {
3793   assert(true);
3794   return "";
3795 }


3798 ////////////////////////////////////////////////////////////////////////////
3799 //
3800 // OperationalTransformation::print
3801 //
3802 ////////////////////////////////////////////////////////////////////////////


3805 helper OperationalTransformation::print(tabs : Integer) : String {
3806   var code : String := "";
3807   self.usedModelType->reject(mt|mt.name = '_INTERMEDIATE')->forEach(mt) {
3808     code := code + printTabs(tabs) + mt.print(0);
3809   };
3810     code := code + '\n' + printTabs(tabs) + 'transformation ' + self.name + '(';
3811     code := code + printArgs(self.modelParameter);

3813     code := code + ');\n';

3815     self.configProperty->forEach(p) {
3816       code := code + '\nconfiguration property ' + p.name + ' : ' + p.eType.name + ';';
```

```
3817      };

3819      code := code + '\n';

3821      self.intermediateClass->forEach(klass) {
3822        code := code + '\nintermediate class ' + klass.name + ' {\n';
3823        klass.eAttributes->forEach(attr) {
3824          code := code + printTabs(tabs + 1) + attr.name + ' : ' + attr.eType.name + ';\n';
3825        };
3826        code := code + '}';
3827      };

3829      code := code + '\n';

3831      self.intermediateProperty->forEach(p) {
3832        var cp := p.oclAsType(ContextualProperty);
3833        code := code + '\nintermediate property ' +
3834          cp.context.name + '::' + cp.name + ' : ' + cp.eType.name
3835        ;
3836        if cp.initExpression <> null then {
3837          code := code + ' = ' + cp.initExpression.print(tabs);
3838        }
3839        endif;
3840        code := code + ';';
3841      };

3843      code := code + '\n';

3845      var set := self.eAllStructuralFeatures - self.configProperty;

3847      set->forEach(p|p.oclIsKindOf(EAttribute)) {
3848        var pa := p.oclAsType(EAttribute);
3849        code := code + '\nproperty ' + pa.name + ' : ' + pa.eType.name;
3850        code := code + ' = ' + pa.eAnnotations.contents->first().oclAsType(ocl::ecore::OCLExpression).print(0) + ';';
3851      };

3853      code := code + '\n';

3855      code := code + '\n' + self.entry.print(tabs);
3856      self.eOperations->forEach(op | op.oclIsKindOf(MappingOperation)) {
3857        code := code + '\n' + op.oclAsType(MappingOperation).print(tabs);
3858      };
3859      self.eOperations->forEach(op | op.oclIsKindOf(Helper)) {
3860        code := code + '\n' + op.oclAsType(Helper).print(tabs);
3861      };
3862    return code;
3863  }


3866  ////////////////////////////////////////////////////////////////////////////
3867  //
3868  // ModelType::print
3869  //
3870  ////////////////////////////////////////////////////////////////////////////

3872  helper ModelType::print(tabs : Integer) : String {
3873    var code : String := 'modeltype ' + self.name;
3874    if (self.conformanceKind != null) then {
3875        code := code + ' "' + self.conformanceKind + '"';
3876    }
3877    endif;
```

```
3878   code := code + ' uses ';
3879   var package := self._metamodel->first();
3880   var meta : String := package.name;
3881   package := package.eSuperPackage;
3882   while (package != null) {
3883     meta := package.name + '::' + meta;
3884     package := package.eSuperPackage;
3885   };
3886   code := code + meta + '(' + self._metamodel->first().nsURI.quotify('\'') + ');' + '\n';
3887   return code;
3888 }


3891 ////////////////////////////////////////////////////////////////////////////
3892 //
3893 // ModelParameter::print
3894 //
3895 ////////////////////////////////////////////////////////////////////////////

3897 helper ModelParameter::print(tabs : Integer) : String {
3898   var code : String := self.kind.repr() + ' ' + self.name + ': ' + self.eType.name;
3899   return code;
3900 }


3903 ////////////////////////////////////////////////////////////////////////////
3904 //
3905 // VarParameter::print
3906 //
3907 ////////////////////////////////////////////////////////////////////////////

3909 helper VarParameter::print(tabs : Integer) : String {
3910   var code : String := self.name + ' : ' + self.eType.name;
3911   return code;
3912 }


3915 ////////////////////////////////////////////////////////////////////////////
3916 //
3917 // EntryOperation::print
3918 //
3919 ////////////////////////////////////////////////////////////////////////////

3921 helper EntryOperation::print(tabs : Integer) : String {
3922   var code : String = printTabs(tabs) + 'main() {\n';
3923   code := code + self.body.print(tabs + 1) + printTabs(tabs) + '}\n';
3924   return code;
3925 }


3928 ////////////////////////////////////////////////////////////////////////////
3929 //
3930 // Helper::print
3931 //
3932 ////////////////////////////////////////////////////////////////////////////

3934 helper Helper::print(tabs : Integer) : String {
3935   var code : String = printTabs(tabs) + 'helper ';
3936   if (self.context != null) then {
3937     code := code + self.context.eType.name + '::';
3938   }
```

```
3939   endif;
3940   code := code + self.name;
3941   code := code + '(' + printArgs(self.eParameters) + ')';
3942   code := code + ' : ' + printArgs(self._result) + ' {\n';
3943   code := code + self.body.print(tabs + 1);
3944   code := code + printTabs(tabs) + '}\n';
3945   return code;
3946 }


3949 ////////////////////////////////////////////////////////////////////////////
3950 //
3951 // Helper::signature
3952 //
3953 ////////////////////////////////////////////////////////////////////////////

3955 helper Helper::signature() : String {
3956   var code : String;
3957   if (self.context != null) then {
3958     code := code + self.context.eType.name + '::';
3959   }
3960   endif;
3961   code := code + self.name;
3962   code := code + '(' + printArgs(self.eParameters) + ')';
3963   return code;
3964 }


3967 ////////////////////////////////////////////////////////////////////////////
3968 //
3969 // MappingParameter::print
3970 //
3971 ////////////////////////////////////////////////////////////////////////////

3973 helper MappingParameter::print(tabs : Integer) : String {
3974   var code : String := self.name + ' : ' + self.eType.name;
3975   return code;
3976 }


3979 ////////////////////////////////////////////////////////////////////////////
3980 //
3981 // MappingOperaion::pring
3982 //
3983 ////////////////////////////////////////////////////////////////////////////

3985 helper MappingOperation::print(tabs : Integer) : String {
3986   var code : String = printTabs(tabs) + 'mapping ' + self.context.eType.name + '::' + self.name;
3987   code := code + '(' + printArgs(self.eParameters) + ')';
3988   code := code + ' : ' + self._result->first().eType.name;
3989   if (self._when->notEmpty()) then {
3990     code := code + '\n' + printTabs(tabs) + 'when { ' + self._when->first().print(0);
3991     var rest := self._when->asSequence()->subSequence(2, self._when->size());
3992     rest->forEach(expr) {
3993       code := code + '; ' + expr.print(0);
3994     };
3995     code := code + ' }';
3996   }
3997   endif;
3998   code := code + '\n' + printTabs(tabs) + '{\n';
3999   code := code + self.body.print(tabs + 1);
```

```
4000   code := code + printTabs(tabs) + '}\n';
4001   return code;
4002 }


4005 ////////////////////////////////////////////////////////////////////////////
4006 //
4007 // OperationBody::print
4008 //
4009 ////////////////////////////////////////////////////////////////////////////

4011 helper OperationBody::print(tabs : Integer) : String {
4012   var code : String := printExpressions(self.content, tabs);
4013   return code;
4014 }


4017 ////////////////////////////////////////////////////////////////////////////
4018 //
4019 // MappingBody::print
4020 //
4021 ////////////////////////////////////////////////////////////////////////////

4023 helper MappingBody::print(tabs : Integer) : String {
4024   var code : String;
4025   var needsPopulation := self.initSection->notEmpty();
4026   if self.initSection->notEmpty() then {
4027     code := printTabs(tabs) + 'init {\n';
4028     code := code + printExpressions(self.initSection, tabs + 1);
4029     code := code + printTabs(tabs) + '}\n';
4030   }
4031   endif;
4032   if needsPopulation then {
4033     code := code + printTabs(tabs) + 'population {\n';
4034     code := code + printExpressions(self.content, tabs + 1);
4035     code := code + printTabs(tabs) + '}\n';
4036   }
4037   else {
4038     code := code + printExpressions(self.content, tabs);
4039   }
4040   endif;
4041   if self.endSection->notEmpty() then {
4042     code := code + printTabs(tabs) + 'end {\n';
4043     code := code + printExpressions(self.endSection,tabs + 1);
4044     code := code + printTabs(tabs) + '}\n';
4045   }
4046   endif;

4048   return code;
4049 }


4052 ////////////////////////////////////////////////////////////////////////////
4053 //
4054 // MappingCallExp::print
4055 //
4056 ////////////////////////////////////////////////////////////////////////////

4058 helper MappingCallExp::print(tabs : Integer) : String {
4059   var code : String := self.source.print(0) + '.' + 'map ';
4060   code := code + self.referredOperation.print(0) + '(' + printArgs(self.argument) + ')';
```

```
4061   return code;
4062 }



4066 ////////////////////////////////////////////////////////////////////////////
4067 //
4068 // ImperativeIterateExp::print
4069 //
4070 ////////////////////////////////////////////////////////////////////////////

4072 helper ImperativeIterateExp::print(tabs : Integer) : String {
4073   var code : String := self.source.print(0) + '->' +
4074               self.name +
4075               '(';
4076   var iter := self.iterator->first().repr();
4077   code := code + iter;
4078   /*
4079   var iter := self.iterator->first().oclAsType(ocl::ecore::Variable);
4080   code := code + iter.name + ' : ' + iter.eType.name;
4081   self.iterator->subOrderedSet(2,self.iterator->size())->forEach(it) {
4082     iter := it.oclAsType(ocl::ecore::Variable);
4083     code := code + ', ' + iter.name + iter.eType.name;
4084   };
4085   */
4086   if (self.body != null) then
4087     code := code + '|' + self.body.print(0)
4088   endif;
4089   if (self.condition != null) then
4090     code := code + '|' + self.condition.print(0)
4091   endif;
4092   code := code + ')';
4093   return code;
4094 }


4097 ////////////////////////////////////////////////////////////////////////////
4098 //
4099 // ImperativeLoopExp::print
4100 //
4101 ////////////////////////////////////////////////////////////////////////////

4103 helper ImperativeLoopExp::print(tabs : Integer) : String {
4104   var code : String := self.source.print(0) + '->' +
4105               self.name +
4106               '(';
4107   var iter := self.iterator->first().repr();
4108   code := code + iter;
4109   /*
4110   var iter := self.iterator->first().oclAsType(ocl::ecore::Variable);
4111   code := code + iter.name + ' : ' + iter.eType.name;
4112   self.iterator->subOrderedSet(2,self.iterator->size())->forEach(it) {
4113     iter := it.oclAsType(ocl::ecore::Variable);
4114     code := code + ', ' + iter.name + iter.eType.name;
4115   };
4116   */
4117   if (self.condition != null) then {
4118     code := code + '|' + self.condition.print(0)
4119   }
4120   endif;
```

```
4122   code := code + ')';

4124   if (self.body != null) then  {
4125     code := code + self.body.print(tabs);
4126   }
4127   endif;

4129   return code;
4130 }




4135 ////////////////////////////////////////////////////////////////////////////
4136 //
4137 // ObjectExp::print
4138 //
4139 ////////////////////////////////////////////////////////////////////////////

4141 helper ObjectExp::print(tabs : Integer) : String {
4142   var code : String := 'object ';
4143   if (self.referredObject <> null and
4144     self.referredObject.name <> null and
4145       self.referredObject.name <> '') then {
4146     code := code + self.referredObject.name + ' : ';
4147   }
4148   endif;
4149   if (self.instantiatedClass <> null and
4150     self.instantiatedClass.name <> null and
4151     self.instantiatedClass.name <> ''
4152     ) then {
4153     code := code  + self.instantiatedClass.name
4154   }
4155   else {
4156     if (self.eType <> null and
4157       self.eType.name <> null and
4158       self.eType.name <> '') then {
4159       code := code  + ' : ' + self.eType.name;
4160     }
4161     endif;
4162   }
4163   endif;
4164   if self.extent <> null then {
4165     code := code + '@' + self.extent.name;
4166   }
4167   endif;
4168   code := code + ' {\n';
4169   code := code + self.body.print(tabs + 1);
4170   code := code + printTabs(tabs) + '}';
4171   return code;
4172 }


4175 ////////////////////////////////////////////////////////////////////////////
4176 //
4177 // SwitchExp::print
4178 //
4179 ////////////////////////////////////////////////////////////////////////////

4181 helper SwitchExp::print(tabs : Integer) : String {
4182   var code : String := 'switch {\n';
```

```
4183   code := code + printExpressions(self.alternativePart, tabs + 1);
4184   if (self.elsePart != null) then {
4185     code := code + '\n' + printTabs(tabs + 1) + 'else ' + self.elsePart.print(tabs + 1);
4186   }
4187   endif;
4188   code := code + '\n' + printTabs(tabs) + '}';
4189   return code;
4190 }


4193 ////////////////////////////////////////////////////////////////////////////
4194 //
4195 // AltExp::print
4196 //
4197 ////////////////////////////////////////////////////////////////////////////

4199 helper AltExp::print(tabs : Integer) : String {
4200   var code : String := 'case ' + ' (' + self.condition.print(0) + ')\n';
4201   code :=  code + printTabs(tabs) + self.body.print(tabs);
4202   return code;
4203 }


4206 ////////////////////////////////////////////////////////////////////////////
4207 //
4208 // AssertExp::print
4209 //
4210 ////////////////////////////////////////////////////////////////////////////

4212 helper AssertExp::print(tabs : Integer) : String {
4213   var code : String := 'assert(' + self.assertion.print(0) + ')';
4214   return code;
4215 }


4218 ////////////////////////////////////////////////////////////////////////////
4219 //
4220 // AssignExp::print
4221 //
4222 ////////////////////////////////////////////////////////////////////////////

4224 helper AssignExp::print(tabs : Integer) : String {
4225   var code : String := self.left.print(0);
4226   code := code  + ' := ';
4227   self.value->forEach(expr) {
4228     code := code  + expr.print(0);
4229   };
4230   return code;
4231 }


4234 ////////////////////////////////////////////////////////////////////////////
4235 //
4236 // BlockExp::print
4237 //
4238 ////////////////////////////////////////////////////////////////////////////

4240 helper BlockExp::print(tabs : Integer) : String {
4241   var code : String := '{\n';
4242   code := code + printExpressions(self.body,tabs + 1);
4243   code := code + printTabs(tabs) + '}';
```

```
4244   return code
4245 }


4248 ////////////////////////////////////////////////////////////////////////////
4249 //
4250 // BreakExp::print
4251 //
4252 ////////////////////////////////////////////////////////////////////////////

4254 helper BreakExp::print(tabs : Integer) : String {
4255   var code : String := 'break';
4256   return code;
4257 }


4260 ////////////////////////////////////////////////////////////////////////////
4261 //
4262 // CatchExp::print
4263 //
4264 ////////////////////////////////////////////////////////////////////////////

4266 helper CatchExp::print(tabs : Integer) : String {
4267   var code : String := 'catch';
4268   return code;
4269 }


4272 ////////////////////////////////////////////////////////////////////////////
4273 //
4274 // ComputeExp::print
4275 //
4276 ////////////////////////////////////////////////////////////////////////////

4278 helper ComputeExp::print(tabs : Integer) : String {
4279   var code : String := 'compute(' + self.returnedElement.repr() + ') ';
4280   code := code + self.body.print(tabs);
4281   return code;
4282 }


4285 ////////////////////////////////////////////////////////////////////////////
4286 //
4287 // ContinueExp::print
4288 //
4289 ////////////////////////////////////////////////////////////////////////////

4291 helper ContinueExp::print(tabs : Integer) : String {
4292   var code : String := 'continue';
4293   return code;
4294 }


4297 ////////////////////////////////////////////////////////////////////////////
4298 //
4299 // InstantiationExp::print
4300 //
4301 ////////////////////////////////////////////////////////////////////////////

4303 helper InstantiationExp::print(tabs : Integer) : String {
4304   var code : String := 'InstantiationExpNotImpl';
```

```
4305   return code;
4306 }


4309 ////////////////////////////////////////////////////////////////////////////
4310 //
4311 // DictLiteralPart::print
4312 //
4313 ////////////////////////////////////////////////////////////////////////////

4315 helper DictLiteralPart::print(tabs : Integer) : String {
4316   var code : String := self.key.print(tabs) + ' = ' + self.value.print(tabs);
4317   return code;
4318 }


4321 ////////////////////////////////////////////////////////////////////////////
4322 //
4323 // DictLiteralExp::print
4324 //
4325 ////////////////////////////////////////////////////////////////////////////

4327 helper DictLiteralExp::print(tabs : Integer) : String {
4328   var code : String := 'Dict { ';
4329   code := code + printArgs(self.part->asOrderedSet());
4330   code := code + ' }';
4331   return code;
4332 }


4335 ////////////////////////////////////////////////////////////////////////////
4336 //
4337 // ListLiteralExp::print
4338 //
4339 ////////////////////////////////////////////////////////////////////////////

4341 helper ListLiteralExp::print(tabs : Integer) : String {
4342   var code : String := 'ListLiteralExpNotImpl';
4343   return code;
4344 }


4347 ////////////////////////////////////////////////////////////////////////////
4348 //
4349 // LogExp::print
4350 //
4351 ////////////////////////////////////////////////////////////////////////////

4353 helper LogExp::print(tabs : Integer) : String {
4354   var code : String := 'log(' + printArgs(self.argument) + ')';
4355   return code;
4356 }



4360 ////////////////////////////////////////////////////////////////////////////
4361 //
4362 // RaiseExp::print
4363 //
4364 ////////////////////////////////////////////////////////////////////////////
```

```
4366 helper RaiseExp::print(tabs : Integer) : String {
4367   var code : String := 'raise';
4368   return code;
4369 }


4372 ////////////////////////////////////////////////////////////////////////////
4373 //
4374 // ReturnExp::print
4375 //
4376 ////////////////////////////////////////////////////////////////////////////

4378 helper ReturnExp::print(tabs : Integer) : String {
4379   var code : String := 'return ' + self.value.print(tabs);
4380   return code;
4381 }



4385 ////////////////////////////////////////////////////////////////////////////
4386 //
4387 // TryExp::print
4388 //
4389 ////////////////////////////////////////////////////////////////////////////

4391 helper TryExp::print(tabs : Integer) : String {
4392   var code : String := 'try';
4393   return code;
4394 }


4397 ////////////////////////////////////////////////////////////////////////////
4398 //
4399 // Typedef::print
4400 //
4401 ////////////////////////////////////////////////////////////////////////////

4403 helper Typedef::print(tabs : Integer) : String {
4404   var code : String := 'typedef';
4405   return code;
4406 }




4411 ////////////////////////////////////////////////////////////////////////////
4412 //
4413 // UnlinkExp::print
4414 //
4415 ////////////////////////////////////////////////////////////////////////////

4417 helper UnlinkExp::print(tabs : Integer) : String {
4418   var code : String := 'unlink';
4419   return code;
4420 }


4423 ////////////////////////////////////////////////////////////////////////////
4424 //
4425 // UnpackExp::print
4426 //
```

```
4427 ////////////////////////////////////////////////////////////////////////////////

4429 helper UnpackExp::print(tabs : Integer) : String {
4430   var code : String := 'unpack';
4431   return code;
4432 }


4435 ////////////////////////////////////////////////////////////////////////////////
4436 //
4437 // VariableInitExp::print
4438 //
4439 ////////////////////////////////////////////////////////////////////////////////

4441 helper VariableInitExp::print(tabs : Integer) : String {
4442   var code := 'var ' + self.referredVariable.name;
4443   if (self.referredVariable.eType != null and
4444     self.referredVariable.eType.name != null and
4445     self.referredVariable.eType.name != '') then {
4446     code := code +  ' : ' + self.referredVariable.eType.name;
4447   }
4448   endif;
4449   if (self.referredVariable.initExpression != null) then {
4450     var iexp := self.referredVariable.initExpression;
4451     code := code + ' := ' + iexp.print(tabs).replace('\n', ' ').replace('    ', ' ');
4452   }
4453   endif;
4454   return code;
4455 }


4458 ////////////////////////////////////////////////////////////////////////////////
4459 //
4460 // WhileExp::print
4461 //
4462 ////////////////////////////////////////////////////////////////////////////////

4464 helper WhileExp::print(tabs : Integer) : String {
4465   var code : String := 'while (' + self.condition.print(0) + ') ';
4466   code := code + self.body.print(tabs);
4467   return code;
4468 }


4471 ////////////////////////////////////////////////////////////////////////////////
4472 //
4473 // CollectionLiteralPart::print
4474 //
4475 ////////////////////////////////////////////////////////////////////////////////

4477 helper ocl::expressions::CollectionLiteralPart::print(tabs : Integer) : String {
4478   return self.repr();
4479 }


4482 ////////////////////////////////////////////////////////////////////////////////
4483 //
4484 // CollectionItem::print
4485 //
4486 ////////////////////////////////////////////////////////////////////////////////
```

```
4488 helper CollectionItem::print(tabs : Integer) : String {
4489   return self.item.print(0);
4490 }


4493 ////////////////////////////////////////////////////////////////////////////
4494 //
4495 // CollectionLiteralExp::print
4496 //
4497 ////////////////////////////////////////////////////////////////////////////

4499 helper CollectionLiteralExp::print(tabs : Integer) : String {
4500   var code : String := '';
4501   switch {
4502     case (self.eType.oclIsTypeOf(SetType) or self.kind = ocl::expressions::CollectionKind::Set)
4503     {
4504       code := code + 'Set{';
4505     }
4506     case (self.eType.oclIsTypeOf(SequenceType) or self.kind = ocl::expressions::CollectionKind::Sequence)
4507     {
4508       code := code + 'Sequence{';
4509     }
4510     case (self.eType.oclIsTypeOf(OrderedSetType) or self.kind = ocl::expressions::CollectionKind::OrderedSet)
4511     {
4512       code := code + 'OrderedSet{';
4513     }
4514     case (self.eType.oclIsTypeOf(BagType) or self.kind = ocl::expressions::CollectionKind::Bag)
4515     {
4516       code := code + 'Bag{';
4517     }
4518     case (self.eType.oclIsTypeOf(ListType))
4519     {
4520       code := code + 'List{'
4521     }
4522   } ;

4524   code := code + printArgs(self.part);

4526   code := code + '}';
4527   return code;
4528 }


4531 ////////////////////////////////////////////////////////////////////////////
4532 //
4533 // TypeExp::print
4534 //
4535 ////////////////////////////////////////////////////////////////////////////

4537 helper ocl::ecore::TypeExp::print(tabs : Integer) : String {
4538   var code : String := self.referredType.print(0);
4539   return code;
4540 }


4543 ////////////////////////////////////////////////////////////////////////////
4544 //
4545 // StringLiteralExp::print
4546 //
4547 ////////////////////////////////////////////////////////////////////////////
```

```
4549 helper ocl::ecore::StringLiteralExp::print(tabs : Integer) : String {
4550   var code : String :=  self.stringSymbol.replace('\n','\\n').replace('\t','\\t').quotify('\'');
4551   return code;
4552 }



4556 ////////////////////////////////////////////////////////////////////////////
4557 //
4558 // OperationCallExp::print
4559 //
4560 ////////////////////////////////////////////////////////////////////////////

4562 helper ocl::ecore::OperationCallExp::print(tabs : Integer) : String {
4563   var opName := self.referredOperation.oclAsType(ecore::EOperation).name;
4564   var code : String := '';
4565   if (self.source <> null) then {
4566     code := code + self.source.print(0);
4567       if (opName = '+' or
4568         opName = '-' or
4569         opName = '/' or
4570         opName = '*' or
4571         opName = 'div' or
4572         opName = 'mod' or
4573         opName = '<' or
4574         opName = '>' or
4575         opName = '>=' or
4576         opName = '<=' or
4577         opName = '<>' or
4578         opName = '=') then {
4579         code := code + ' ' + opName + ' ' + printArgs(self.argument);
4580         return code;
4581       }
4582       endif;
4583       var srcType := self.source.getType();
4584       if (srcType <> null and (srcType.oclIsKindOf(ocl::ecore::CollectionType) or srcType.oclIsKindOf(DictionaryType)))
             then {
4585         code := code + '->';
4586       }
4587       else {
4588         code := code + '.';
4589       }
4590       endif;
4591   }
4592   endif;

4594   code := code + opName + '(';
4595   code := code + printArgs(self.argument).replace('\n', ' ') + ')';

4597   return code;
4598 }


4601 ////////////////////////////////////////////////////////////////////////////
4602 //
4603 // PropertyCallExp::print
4604 //
4605 ////////////////////////////////////////////////////////////////////////////

4607 helper ocl::ecore::PropertyCallExp::print(tabs : Integer) : String {
4608   var code : String := self.source.print(0);
```

```
4609   code := code + '.' + self.referredProperty.print(0);

4611   //Sequence{self.referredProperty}->switch(p) {
4612   //  case (p.oclIsKindOf(EAttribute)) {code := code + '.' + self.referredProperty.oclAsType(EAttribute).name}
4613   //  case (p.oclIsKindOf(EReference)) {code := code + '.' + self.referredProperty.oclAsType(EReference).name}
4614   //  else {}
4615   //};

4617   return code;
4618 }


4621 ////////////////////////////////////////////////////////////////////////////
4622 //
4623 // IfExp::print
4624 //
4625 ////////////////////////////////////////////////////////////////////////////

4627 helper ocl::ecore::IfExp::print(tabs : Integer) : String {
4628   var code : String := 'if ' + '(' + self.condition.print(0) + ')' + ' then ';
4629   code := code + self.thenExpression.print(tabs) + '\n';
4630   if self.elseExpression != null then {
4631     code := code + printTabs(tabs) + 'else ' + self.elseExpression.print(tabs) + '\n';
4632   }
4633   endif;
4634   code := code + printTabs(tabs) + 'endif';
4635   return code;
4636 }


4639 ////////////////////////////////////////////////////////////////////////////
4640 //
4641 // OCLExpression::print
4642 //
4643 ////////////////////////////////////////////////////////////////////////////

4645 helper ocl::ecore::OCLExpression::print(tabs : Integer) : String {
4646   var code : String := self.repr();
4647   return code;
4648 }


4651 ////////////////////////////////////////////////////////////////////////////
4652 //
4653 // OCLExpression::print
4654 //
4655 ////////////////////////////////////////////////////////////////////////////

4657 helper ocl::expressions::OCLExpression::print(tabs : Integer) : String {
4658   var code : String := self.repr();
4659   return code;
4660 }



4664 ////////////////////////////// QvtBTA ////////////////////////////////////////////////

4666 intermediate class TransformationContext
4667 {
4668   fixedElements : OrderedSet(EObject);
4669   varElements : OrderedSet(EObject);
```

```
4670   varClasses : OrderedSet(EClass);
4671   inModel : ModelParameter;
4672   outModel : ModelParameter;
4673 };

4675 constructor TransformationContext::TransformationContext() {

4677 }


4680 ////////////////////////////////////////////////////////////////////////////
4681 //
4682 // createTransformationContext
4683 //
4684 ////////////////////////////////////////////////////////////////////////////

4686 helper createTransformationContext() : TransformationContext {
4687   return new TransformationContext();
4688 }


4691 ////////////////////////////////////////////////////////////////////////////
4692 //
4693 // OperationalTransformation::bta
4694 //
4695 ////////////////////////////////////////////////////////////////////////////

4697 helper OperationalTransformation::bta(bt : Dict(String, BTAKind), inout context : TransformationContext) : BTAKind {
4698   var inModel : ModelParameter := self.modelParameter->select(mp|mp.kind = DirectionKind::_in)->asSequence()->first();
4699   var outModel : ModelParameter := self.modelParameter->select(mp|mp.kind = DirectionKind::_out)->asSequence()->first();
4700   context.inModel := inModel;
4701   context.outModel := outModel;

4703   var inModelType : EPackage := inModel.eType.oclAsType(ModelType)._metamodel->first();
4704   var outModelType := outModel.eType.oclAsType(ModelType)._metamodel->first();

4706   var annotations := inModelType.eAnnotations;
4707   var fixedAnnot := annotations->select(ann|ann.source = 'FIXED')->asSequence()->first();
4708   var varAnnot := annotations->select(ann|ann.source = 'VAR')->asSequence()->first();
4709   context.fixedElements := fixedAnnot._references;

4711   varAnnot._references->forEach(element) {
4712     switch {
4713       case (element.oclIsTypeOf(EClass))
4714       {
4715         context.varClasses += element.oclAsType(EClass);
4716         break;
4717       }
4718       case (element.oclIsTypeOf(EReference))
4719       {
4720         var ref := element.oclAsType(EReference);
4721         context.varClasses += ref.eContainingClass;
4722       }
4723       case (element.oclIsTypeOf(EAttribute))
4724       {
4725         var attr := element.oclAsType(EAttribute);
4726         context.varClasses += attr.eContainingClass;
4727       }
4728     };

4730   };
```

```
4732   var statements := self.entry.body.content;
4733   var transBTA := BTAKind::STATIC;
4734   statements->forEach(expr) {
4735     if (expr.bta(bt, context) = BTAKind::DYNAMIC) then {
4736       transBTA := BTAKind::DYNAMIC;
4737     }
4738     endif;
4739   };

4741   return transBTA;
4742 }



4746 ////////////////////////////////////////////////////////////////////////////
4747 //
4748 // ASTNode::bta
4749 //
4750 ////////////////////////////////////////////////////////////////////////////

4752 helper ASTNode::bta(bt : Dict(String, BTAKind), inout context : TransformationContext) : BTAKind {
4753   return BTAKind::DYNAMIC;
4754 }


4757 ////////////////////////////////////////////////////////////////////////////
4758 //
4759 // PropertyCallExp::bta
4760 //
4761 ////////////////////////////////////////////////////////////////////////////

4763 helper PropertyCallExp::bta(bt : Dict(String, BTAKind), inout context : TransformationContext) : BTAKind {
4764   var srcBT := self.source.bta(bt, context);
4765   if srcBT = BTAKind::STATIC then {
4766     return BTAKind::STATIC;
4767   }
4768   endif;
4769   return BTAKind::MAYBE;
4770 }


4773 ////////////////////////////////////////////////////////////////////////////
4774 //
4775 // LiteralExp::bta
4776 //
4777 ////////////////////////////////////////////////////////////////////////////

4779 helper LiteralExp::bta(bt : Dict(String, BTAKind), inout context : TransformationContext) : BTAKind {
4780   return BTAKind::STATIC;
4781 }


4784 ////////////////////////////////////////////////////////////////////////////
4785 //
4786 // LogExp::bta
4787 //
4788 ////////////////////////////////////////////////////////////////////////////

4790 helper LogExp::bta(bt : Dict(String, BTAKind), inout context : TransformationContext) : BTAKind {
4791   return BTAKind::DYNAMIC;
```

```
4792 }


4795 ////////////////////////////////////////////////////////////////////////////
4796 //
4797 // ReturnExp::bta
4798 //
4799 ////////////////////////////////////////////////////////////////////////////

4801 helper ReturnExp::bta(bt : Dict(String, BTAKind), inout context : TransformationContext) : BTAKind {
4802   return self.value.bta(bt, context);
4803 }


4806 ////////////////////////////////////////////////////////////////////////////
4807 //
4808 // AssignExp::bta
4809 //
4810 ////////////////////////////////////////////////////////////////////////////

4812 helper AssignExp::bta(bt : Dict(String, BTAKind), inout context : TransformationContext) : BTAKind {
4813   var valueBT := self.value->first().bta(bt, context);
4814   if valueBT <> BTAKind::STATIC then {
4815     return valueBT;
4816   }
4817   endif;
4818   return self.left.bta(bt, context);
4819 }


4822 ////////////////////////////////////////////////////////////////////////////
4823 //
4824 // VariableExp::bta
4825 //
4826 ////////////////////////////////////////////////////////////////////////////

4828 helper VariableExp::bta(bt : Dict(String, BTAKind), inout context : TransformationContext) : BTAKind {
4829   if (bt->hasKey(self.name)) then {
4830     return bt->get(self.name);
4831   }
4832   endif;
4833   return BTAKind::MAYBE;
4834 }


4837 ////////////////////////////////////////////////////////////////////////////
4838 //
4839 // VariableInitExp::bta
4840 //
4841 ////////////////////////////////////////////////////////////////////////////

4843 helper VariableInitExp::bta(bt : Dict(String, BTAKind), inout context : TransformationContext) : BTAKind {
4844   var varName := self.referredVariable.name;
4845   var staticReferredVars :=
4846     self.referredVariable.initExpression.
4847       allSubobjectsOfType(VariableExp)->
4848         oclAsType(VariableExp)->
4849           select(v|bt->get(v.name) = BTAKind::STATIC)
4850   ;
4851   var dynamicReferredVars :=
4852     self.referredVariable.initExpression.allSubobjectsOfType(VariableExp)->
```

```
4853       oclAsType(VariableExp)->select(v|bt->get(v.name) = BTAKind::DYNAMIC)
4854   ;
4855   var initBTA := self.referredVariable.initExpression.bta(bt, context);
4856   bt->put(varName, initBTA);
4857   return initBTA;

4859 //  if (dynamicReferredVars->size() > 0 ) then {
4860 //    bt->put(varName, BTAKind::DYNAMIC);
4861 //    return BTAKind::DYNAMIC;
4862 //  }
4863 //  else {
4864 //    bt->put(varName, BTAKind::STATIC);
4865 //    return BTAKind::STATIC;
4866 //  }
4867 //  endif;
4868 //  return BTAKind::STATIC;

4870 }


4873 ////////////////////////////////////////////////////////////////////////////
4874 //
4875 // ImperativeIterateExp::bta
4876 //
4877 ////////////////////////////////////////////////////////////////////////////

4879 helper ImperativeIterateExp::bta(bt : Dict(String, BTAKind), inout context : TransformationContext) : BTAKind {
4880   return self.source.bta(bt, context);
4881 }


4884 ////////////////////////////////////////////////////////////////////////////
4885 //
4886 // OperationCallExp::bta
4887 //
4888 ////////////////////////////////////////////////////////////////////////////

4890 helper OperationCallExp::bta(bt : Dict(String, BTAKind), inout context : TransformationContext) : BTAKind {
4891   if (self.source.oclIsTypeOf(VariableExp) and
4892     self.source.oclAsType(VariableExp).name = context.inModel.name) then {
4893     var modelRef := self.source.oclAsType(VariableExp);
4894     var op := self.referredOperation.oclAsType(EOperation);
4895     if (op.name = 'objectsOfType') then {
4896       var type := self.argument->first().oclAsType(TypeExp).eType.name;
4897       if (context.varClasses->exists(cl|cl.name = type)) then {
4898         return BTAKind::DYNAMIC;
4899       }
4900       else {
4901         // CHECK this later: if a type in the input metamodel is not in the var classes
4902         // can it be declared as STATIC in all situations?
4903         return BTAKind::STATIC;
4904       }
4905       endif;
4906     }
4907     endif;
4908   }
4909   endif;
4910   var srcBTA : BTAKind;
4911   if (self.source.oclIsTypeOf(VariableExp) and
4912     self.source.oclAsType(VariableExp).name = 'this') then {
4913     srcBTA := BTAKind::STATIC;
```

```
4914   }
4915   else {
4916     srcBTA :=  self.source.bta(bt, context);
4917   }
4918   endif;
4919   var callArgs := self.argument;
4920   var callArgsBT := callArgs->bta(bt, context);
4921   var anyNonStaticArg := callArgsBT->exists(t|t <> BTAKind::STATIC);
4922   if (srcBTA <> BTAKind::STATIC or anyNonStaticArg) then {
4923     return BTAKind::MAYBE;
4924   }
4925   endif;

4927   if not self.referredOperation.oclIsKindOf(Helper) then {
4928     return BTAKind::STATIC;
4929   }
4930   endif;

4932   var helperOp := self.referredOperation.oclAsType(Helper);
4933   if bt->hasKey(helperOp.signature()) then {
4934     return BTAKind::STATIC;
4935   }
4936   endif;
4937   var params := helperOp.eParameters->asOrderedSet();

4939   var newBt : Dict(String, BTAKind) := Dict{};
4940   newBt->put(helperOp.signature(), BTAKind::MAYBE);
4941   bt->keys()->forEach(k) {
4942     newBt->put(k, bt->get(k));
4943   };
4944   var i := 1;
4945   var n := params->size();
4946   while (i <= n) {
4947     newBt->put(params->at(i).name, callArgsBT->at(i));
4948     i := i + 1;
4949   };

4951   newBt->put('self', srcBTA);

4953   var bodyBT := self.referredOperation.oclAsType(Helper).body.content->bta(newBt, context);

4955   if bodyBT->exists(b|b <> BTAKind::STATIC) then {
4956     return BTAKind::MAYBE;
4957   }
4958   endif;
4959   return BTAKind::STATIC;
4960 }


4963 ////////////////////////////////////////////////////////////////////////////
4964 //
4965 // ConstructorBody::bta
4966 //
4967 ////////////////////////////////////////////////////////////////////////////

4969 helper ConstructorBody::bta(bt : Dict(String, BTAKind), inout context : TransformationContext) : BTAKind {
4970   if (self.content->bta(bt, context)->exists(x| x <> BTAKind::STATIC)) then {
4971     return BTAKind::MAYBE;
4972   }
4973   endif;
4974   return BTAKind::STATIC;
```

```
4975 }


4978 ////////////////////////////////////////////////////////////////////////////
4979 //
4980 // ObjectExp::bta
4981 //
4982 ////////////////////////////////////////////////////////////////////////////

4984 helper ObjectExp::bta(bt : Dict(String, BTAKind), inout context : TransformationContext) : BTAKind {
4985   return self.body.bta(bt, context);
4986 }


4989 ////////////////////////////////////////////////////////////////////////////
4990 //
4991 // IfExp::bta
4992 //
4993 ////////////////////////////////////////////////////////////////////////////

4995 helper IfExp::bta(bt : Dict(String, BTAKind), inout context : TransformationContext) : BTAKind {
4996   var conditionBT := self.condition.bta(bt, context);
4997   var thenBT := self.thenExpression.bta(bt, context);
4998   var elseBT := if self.elseExpression <> null then self.elseExpression.bta(bt, context) else BTAKind::STATIC endif;
4999   if (thenBT = BTAKind::STATIC and elseBT = BTAKind::STATIC) then {
5000     return BTAKind::STATIC;
5001   }
5002   endif;
5003   if (conditionBT = BTAKind::STATIC) then {
5004     // if we could eval the condition
5005     return BTAKind::MAYBE;
5006   }
5007   endif;
5008   return BTAKind::DYNAMIC;
5009 }


5012 ////////////////////////////////////////////////////////////////////////////
5013 //
5014 // ForExp::bta
5015 //
5016 ////////////////////////////////////////////////////////////////////////////

5018 helper ForExp::bta(bt : Dict(String, BTAKind), inout context : TransformationContext) : BTAKind {
5019   var srcBT := self.source.bta(bt, context);
5020   var itername := self.iterator->first().oclAsType(Variable).name;
5021   if srcBT = BTAKind::STATIC then {
5022     var newBt : Dict(String, BTAKind) := Dict{};
5023     bt->keys()->forEach(k) {
5024       newBt->put(k, bt->get(k));
5025     };
5026     newBt->put(itername, srcBT);
5027     var conditionBT :=
5028       if (self.condition <> null) then
5029         self.condition.bta(newBt, context)
5030       else
5031         BTAKind::STATIC
5032       endif
5033     ;
5034     if conditionBT = BTAKind::STATIC then {
5035       var bodyBT := self.body.bta(newBt, context);
```

```
5036       return bodyBT;
5037     }
5038     endif;

5040   }
5041   endif;
5042   return srcBT;
5043 }


5046 ////////////////////////////////////////////////////////////////////////////
5047 //
5048 // WhileExp::bta
5049 //
5050 ////////////////////////////////////////////////////////////////////////////

5052 helper WhileExp::bta(bt : Dict(String, BTAKind), inout context : TransformationContext) : BTAKind {
5053   var condBT := self.condition.bta(bt, context);
5054   if (condBT = BTAKind::STATIC) then {
5055     return self.body.bta(bt, context);
5056   }
5057   endif;
5058   return condBT;
5059 }


5062 ////////////////////////////////////////////////////////////////////////////
5063 //
5064 // BlockExp::bta
5065 //
5066 ////////////////////////////////////////////////////////////////////////////

5068 helper BlockExp::bta(bt : Dict(String, BTAKind), inout context : TransformationContext) : BTAKind {
5069   var bodyBT := self.body->bta(bt, context);
5070   var blockBT : BTAKind;

5072   if bodyBT->exists(b|b = BTAKind::DYNAMIC) then {
5073     blockBT := BTAKind::DYNAMIC
5074   }
5075   else {
5076     if bodyBT->exists(b|b = BTAKind::MAYBE) then {
5077       blockBT := BTAKind::MAYBE;
5078     }
5079     else {
5080       blockBT := BTAKind::STATIC;
5081     }
5082     endif;
5083   }
5084   endif;
5085   return blockBT;
5086 }

5088 ////////////////////////////////////// QvtMix ////////////////////////////////////////////////
5089 property bindingTimes : Dict(String, BTAKind) = Dict{};
5090 property environment  : Environment = new Environment();
5091 property context : TransformationContext = new TransformationContext();

5093 property newOperations : OrderedSet(EOperation) = OrderedSet{};
5094 property memoizeTables : Dict(String, EAttribute) = Dict{};

5096 property tempCount : Integer = 0;
```

```
property mixCount : Dict(String, Integer) = Dict{};



////////////////////////////////////////////////////////////////////////////
//
// ImperativeOperation::newMix
//
////////////////////////////////////////////////////////////////////////////

helper ImperativeOperation::newMix() : String {
  var res : String := self.name + '__mix';
  if mixCount->hasKey(self.name) then {
    var n := mixCount->get(self.name);
    n := n + 1;
    mixCount->put(self.name, n);
    res := res + n.print();
  }
  else {
    mixCount->put(self.name, 1);
    res := res + '1';
  }
  endif;
  return res;
}



////////////////////////////////////////////////////////////////////////////
//
// MappingOperation::mappingMixName
//
////////////////////////////////////////////////////////////////////////////

helper MappingOperation::mappingMixName() : String {
  var res : String := self.name + '__mix';
  if not mixCount->hasKey(self.name) then {
    mixCount->put(self.name, 1);
  }
  endif;
  return res;
}



////////////////////////////////////////////////////////////////////////////
//
// newTemp
//
////////////////////////////////////////////////////////////////////////////

helper newTemp() : String {
  tempCount := tempCount + 1;
  return '__temp_' + tempCount.print();
}



////////////////////////////////////////////////////////////////////////////
//
// OperationalTransformation::mix
//
```

```
5158 ////////////////////////////////////////////////////////////////////////////

5160 helper OperationalTransformation::mix() : OperationalTransformation {
5161   environment := createEnvironment();
5162   //context := createTransformationContext();
5163   self.modelParameter->forEach(m) {
5164     bindingTimes->put(m.name, BTAKind::DYNAMIC);
5165     switch {
5166       case (m.kind = DirectionKind::_in)
5167         {
5168           environment.put(m.name, createFrame(inputModel));
5169         }
5170       case (m.kind = DirectionKind::_out)
5171         {
5172           environment.put(m.name, createFrame(outputModel));
5173         }
5174     }
5175   };
5176   var ops := self.eOperations;
5177   ops := ops->reject(o|o.oclIsKindOf(EntryOperation));

5179   var res := object OperationalTransformation {
5180       name := self.name;
5181       eStructuralFeatures := self.eStructuralFeatures;
5182       modelParameter := self.modelParameter;
5183       moduleImport := self.moduleImport;
5184       usedModelType := self.usedModelType;
5185       eOperations := ops;
5186       intermediateClass := self.intermediateClass;
5187       intermediateProperty := self.intermediateProperty;
5188       configProperty := self.configProperty;

5190       var newEntry := self.entry.mix(environment, bindingTimes);
5191       entry := newEntry;
5192       eOperations += newEntry;
5193       eStructuralFeatures += memoizeTables->values();
5194     };
5195     newOperations->forEach(o) {
5196       res.eOperations += o;
5197     };

5199     return res;
5200 }




5205 ////////////////////////////////////////////////////////////////////////////
5206 //
5207 // EntryOperation::mix
5208 //
5209 ////////////////////////////////////////////////////////////////////////////

5211 helper EntryOperation::mix(env : OclAny, bt : Dict(String, BTAKind)) : EntryOperation {

5213   tempCount := 0;
5214   var ret : EntryOperation := object EntryOperation {
5215     name := self.name;
5216     context := self.context;
5217     body := self.body.mix(env, bt);
5218   };
```

```
5219   return ret;
5220 }


5223 ////////////////////////////////////////////////////////////////////////////////
5224 //
5225 // Helper::mix
5226 //
5227 ////////////////////////////////////////////////////////////////////////////////

5229 helper Helper::mix(env : OclAny, bt : Dict(String, BTAKind)) : Helper {
5230   var oldTempCount := tempCount;
5231   tempCount := 0;
5232   var resOp := self.deepclone().oclAsType(Helper);
5233   resOp.name := resOp.newMix();
5234   resOp.body := resOp.body.mix(env, bt);
5235   tempCount := oldTempCount;
5236   return resOp;
5237 }


5240 ////////////////////////////////////////////////////////////////////////////////
5241 //
5242 // MappingBody::mix
5243 //
5244 ////////////////////////////////////////////////////////////////////////////////

5246 helper MappingBody::mix(env : OclAny, bt : Dict(String, BTAKind)) : OperationBody {
5247   var mixInit : OrderedSet(OCLExpression) := OrderedSet{};
5248   var originalInit := self.initSection->deepclone().oclAsType(OCLExpression);
5249   originalInit->forEach(expr) {
5250     var bindingTime := expr.onlineBta(env, bt);
5251     if (bindingTime = BTAKind::DYNAMIC) then {
5252       mixInit += expr.mixReduce(env, bt);
5253     }
5254     else {
5255       mixInit += expr.reduce(env, bt);
5256     }
5257     endif;
5258   };
5259   var mixPopulation : OrderedSet(OCLExpression) := OrderedSet{};
5260   var originalPopulation := self.content->deepclone()->oclAsType(OCLExpression);
5261   originalPopulation->forEach(expr) {
5262     var bindingTime := expr.onlineBta(env, bt);
5263     if (bindingTime = BTAKind::DYNAMIC) then {
5264       mixPopulation += expr.polyMixReduce(env, bt, self.operation);
5265     }
5266     else {
5267       mixPopulation += expr.reduce(env, bt);
5268     }
5269     endif;
5270   };
5271   var mixBody : MappingBody := new MappingBody();
5272   mixBody.content := mixPopulation;
5273   mixBody.initSection := mixInit;

5275   return mixBody;
5276 }


5279 ////////////////////////////////////////////////////////////////////////////////
```

```
//
// MappingOperation::mix
//
////////////////////////////////////////////////////////////////////////////

helper MappingOperation::mix(env : OclAny, bt : Dict(String, BTAKind)) : MappingOperation {
  var oldTempCount := tempCount;
  tempCount := 0;
  var ret : MappingOperation := self.deepclone().oclAsType(MappingOperation);
  ret.name := ret.mappingMixName();
  ret.body := ret.body.mix(env, bt);
  ret.eParameters := self.eParameters->deepclone()->oclAsType(EParameter);
  tempCount := oldTempCount;
  return ret;
}



////////////////////////////////////////////////////////////////////////////
//
// OperationBody::mix
//
////////////////////////////////////////////////////////////////////////////

helper OperationBody::mix(env : OclAny, bt : Dict(String, BTAKind)) : OperationBody {
    var expressions : OrderedSet(OCLExpression) := OrderedSet{};
    var originalExpr := self.content->deepclone()->oclAsType(OCLExpression);
    originalExpr->forEach(expr) {
      var bindingTime := expr.onlineBta(env, bt);
      if (bindingTime = BTAKind::DYNAMIC) then {
        expressions += expr.mixReduce(env, bt);
      }
      else {
        expressions += expr.reduce(env, bt);
      }
      endif;
      if expressions->last().oclIsTypeOf(ReturnExp) then {
        break;
      }
      endif;
    };
    var mixBody : OperationBody := object OperationBody {
      content := expressions;
    };
    return mixBody;
}


////////////////////////////////////////////////////////////////////////////
//
// ASTNode::onlineBta
//
////////////////////////////////////////////////////////////////////////////

helper ASTNode::onlineBta(env : OclAny, bt : Dict(String, BTAKind)) : BTAKind {
  return BTAKind::STATIC;
}


////////////////////////////////////////////////////////////////////////////
//
```

```
// OCLExpression::onlineBta
//
////////////////////////////////////////////////////////////////////////////////

helper OCLExpression::onlineBta(env : OclAny, bt : Dict(String, BTAKind)) : BTAKind {
  return
    if self.bta(bt, context) = BTAKind::STATIC then
      BTAKind::STATIC
    else
      BTAKind::DYNAMIC
    endif
  ;
}


////////////////////////////////////////////////////////////////////////////////
//
// VariableExp::onlineBta
//
////////////////////////////////////////////////////////////////////////////////

helper VariableExp::onlineBta(env : OclAny, bt : Dict(String, BTAKind)) : BTAKind {
  var offBt := self.bta(bt, context);
  if offBt = BTAKind::MAYBE then {
    if getEnvironment(env).hasKey(self.name) then {
      return BTAKind::STATIC;
    }
    else {
      return BTAKind::DYNAMIC;
    }
    endif;
  }
  endif;
  return offBt;
}


////////////////////////////////////////////////////////////////////////////////
//
// VariableInitExp::onlineBta
//
////////////////////////////////////////////////////////////////////////////////

helper VariableInitExp::onlineBta(env : OclAny, bt : Dict(String, BTAKind)) : BTAKind {
  return self.referredVariable.initExpression.onlineBta(env, bt);
}


////////////////////////////////////////////////////////////////////////////////
//
// IfExp::onlineBta
//
////////////////////////////////////////////////////////////////////////////////

helper IfExp::onlineBta(env : OclAny, bt : Dict(String, BTAKind)) : BTAKind {
  var conditionBt := self.condition.onlineBta(env, bt);
  if conditionBt = BTAKind::STATIC then {
    var condition : Boolean := self.condition.eval(getEnvironment(env)).oclAsType(Boolean);
    if condition then {
      return self.thenExpression.onlineBta(env, bt);
    }
```

```
5402     else {
5403       return self.elseExpression.onlineBta(env, bt);
5404     }
5405     endif;
5406   }
5407   endif;

5409   var thenBt := self.thenExpression.onlineBta(env, bt);
5410   var elseBt := self.elseExpression.onlineBta(env, bt);
5411   if thenBt = BTAKind::STATIC and elseBt = BTAKind::STATIC then {
5412     var thenVal := self.thenExpression.eval(getEnvironment(env));
5413     var elseVal := self.elseExpression.eval(getEnvironment(env));
5414     return
5415         if elseVal = thenVal then
5416           BTAKind::STATIC
5417         else
5418           BTAKind::DYNAMIC
5419         endif
5420     ;
5421   }
5422   endif;
5423   return BTAKind::DYNAMIC;
5424 }


5427 ////////////////////////////////////////////////////////////////////////////
5428 //
5429 // OperationCallExp::onlineBta
5430 //
5431 ////////////////////////////////////////////////////////////////////////////

5433 helper OperationCallExp::onlineBta(env : OclAny, bt : Dict(String, BTAKind)) : BTAKind {
5434   var offBta := self.bta(bt, context);
5435   if offBta = BTAKind::MAYBE then {
5436     if self.referredOperation.oclIsKindOf(Helper) then {
5437       var newBt : Dict(String, BTAKind) := Dict{};
5438       bt->keys()->forEach(k) {
5439         newBt->put(k, bt->get(k));
5440       };
5441       var params := self.referredOperation.oclAsType(Helper).eParameters->asOrderedSet();
5442       var callArgs := self.argument;

5444       var i := 1;
5445       var n := params->size();
5446       while (i <= n) {
5447         newBt->put(params->at(i).name, callArgs->at(i).onlineBta(env, bt));
5448         i := i + 1;
5449       };

5451       newBt->put('self', self.source.onlineBta(env, bt));

5453       var bodyBT := self.referredOperation.oclAsType(Helper).body.content->bta(newBt, context);

5455       if bodyBT->exists(b|b <> BTAKind::STATIC) then {
5456         return BTAKind::DYNAMIC;
5457       }
5458       endif;
5459       return BTAKind::STATIC;
5460     }
5461     else {
5462       return BTAKind::DYNAMIC;
```

```
5463      }
5464      endif;
5465    }
5466    endif;
5467    return offBta;
5468  }


5471  ////////////////////////////////////////////////////////////////////////////////
5472  //
5473  // ASTNode::eval
5474  //
5475  ////////////////////////////////////////////////////////////////////////////////

5477  helper ASTNode::eval(env : OclAny, bt : Dict(String, BTAKind)) : OclAny {
5478    return null;
5479  }


5482  ////////////////////////////////////////////////////////////////////////////////
5483  //
5484  // OCLExpression::eval
5485  //
5486  ////////////////////////////////////////////////////////////////////////////////

5488  helper OCLExpression::eval(env : OclAny, bt : Dict(String, BTAKind)) : OclAny {
5489    return self.eval(getEnvironment(env));
5490  }



5494  ////////////////////////////////////////////////////////////////////////////////
5495  //
5496  // ASTNode::sideEffectsEval
5497  //
5498  ////////////////////////////////////////////////////////////////////////////////

5500  helper ASTNode::sideEffectsEval(env : OclAny, bt : Dict(String, BTAKind), inout effects : EObject) : OclAny {
5501    return self.eval(env, bt);
5502  }


5505  ////////////////////////////////////////////////////////////////////////////////
5506  //
5507  // WhileExp::sideEffectsEval
5508  //
5509  ////////////////////////////////////////////////////////////////////////////////

5511  helper WhileExp::sideEffectsEval(env : OclAny, bt : Dict(String, BTAKind), inout effects : EObject) : OclAny {
5512    var oldEnv := getEnvironment(env);
5513    var newEnv := oldEnv.copy();
5514    var body :=
5515      if (self.body.oclIsKindOf(BlockExp)) then
5516        self.body.oclAsType(BlockExp).body
5517      else
5518        OrderedSet{self.body}
5519      endif
5520    ;
5521    var vars := body[AssignExp]->
5522      xcollect(exp|exp.left)->
5523        xselect(o|o.oclIsKindOf(VariableExp))->
```

```
5524          oclAsType(VariableExp)->
5525            xcollect(v|v.referredVariable.oclAsType(Variable).name)
5526    ;

5528    self.eval(newEnv);
5529    var assignments : OrderedSet(OclAny) := OrderedSet{};

5531    vars->forEach(v) {
5532      var newVal := newEnv.get(v).value();
5533      var oldVal := oldEnv.get(v).value();
5534      if (newVal <> oldVal) then {
5535        assignments += object AssignExp {
5536          left := object VariableExp {
5537            referredVariable := object Variable {
5538              name := v;
5539            };
5540            name := referredVariable.oclAsType(Variable).name;
5541          };
5542          value := newVal.makeExp();
5543        };
5544      }
5545      endif;
5546    };

5548    setCollectionWrapper(effects, assignments);
5549    return null;
5550  }


5553  ////////////////////////////////////////////////////////////////////////////
5554  //
5555  // ForExp::sideEffectsEval
5556  //
5557  ////////////////////////////////////////////////////////////////////////////

5559  helper ForExp::sideEffectsEval(env : OclAny, bt : Dict(String, BTAKind), inout effects : EObject) : OclAny {
5560    var oldEnv := getEnvironment(env);
5561    var newEnv := oldEnv.copy();
5562    var body :=
5563      if (self.body.oclIsKindOf(BlockExp)) then
5564        self.body.oclAsType(BlockExp).body
5565      else
5566        OrderedSet{self.body}
5567      endif
5568    ;

5570    var vars := body[AssignExp]->
5571      xcollect(exp|exp.left)->
5572        xselect(o|o.oclIsKindOf(VariableExp))->
5573          oclAsType(VariableExp)->xcollect(v|v.referredVariable.oclAsType(Variable).name)
5574    ;
5575    var res := self.eval(newEnv);

5577    var assignments : OrderedSet(OclAny) := OrderedSet{};
5578    vars->forEach(v) {
5579      var newVal := newEnv.get(v).value();
5580      var oldVal := oldEnv.get(v).value();
5581      if (newVal <> oldVal) then {
5582        assignments += object AssignExp {
5583          left := object VariableExp {
5584            referredVariable := object Variable {
```

```
5585              name := v;
5586            };
5587            name := referredVariable.oclAsType(Variable).name;
5588          };
5589          value := newVal.makeExp();
5590        };
5591      }
5592      endif;
5593    };

5595    setCollectionWrapper(effects, assignments);

5597    return res;
5598  }


5601  ////////////////////////////////////////////////////////////////////////////
5602  //
5603  // IfExp::eval
5604  //
5605  ////////////////////////////////////////////////////////////////////////////

5607  helper IfExp::eval(env : OclAny, bt : Dict(String, BTAKind)) : OclAny {
5608    if (self.condition.onlineBta(env, bt) = BTAKind::STATIC) then {
5609      if self.condition.eval(getEnvironment(env)).oclAsType(Boolean) then {
5610        if self.thenExpression.onlineBta(env, bt) = BTAKind::STATIC then {
5611          return self.thenExpression.eval(env, bt);
5612        }
5613        else {
5614          return self.thenExpression;
5615        }
5616        endif;
5617      }
5618      else {
5619        if self.elseExpression.onlineBta(env, bt) = BTAKind::STATIC then {
5620          return self.elseExpression.eval(env, bt);
5621        }
5622        else {
5623          return self.elseExpression;
5624        }
5625        endif;
5626      }
5627      endif;
5628    }
5629    endif;
5630    if (self.thenExpression.onlineBta(env, bt) = BTAKind::STATIC and
5631      self.elseExpression.onlineBta(env, bt) = BTAKind::STATIC) then {
5632      var thenVal := self.thenExpression.eval(env, bt);
5633      var elseVal := self.elseExpression.eval(env, bt);
5634      if (elseVal = thenVal) then {
5635        return thenVal;
5636      }
5637      endif;
5638    }
5639    endif;
5640    return self;

5642  }


5645  ////////////////////////////////////////////////////////////////////////////
```

```
5646 //
5647 // ASTNode::reduce
5648 //
5649 ////////////////////////////////////////////////////////////////////////////////

5651 helper ASTNode::reduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
5652   return null;
5653 }


5656 ////////////////////////////////////////////////////////////////////////////////
5657 //
5658 // OCLExpression::reduce
5659 //
5660 ////////////////////////////////////////////////////////////////////////////////

5662 helper OCLExpression::reduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
5663   var rewrite : OrderedSet(OCLExpression) := OrderedSet{self};
5664   return rewrite;
5665 }


5668 ////////////////////////////////////////////////////////////////////////////////
5669 //
5670 // WhileExp::reduce
5671 //
5672 ////////////////////////////////////////////////////////////////////////////////

5674 helper WhileExp::reduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
5675   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
5676   var sideEffects := createCollectionWrapper(rewrite);

5678   self.sideEffectsEval(env, bt, sideEffects);
5679   sideEffects.collection()->forEach(e) {
5680     rewrite += e.oclAsType(OCLExpression);
5681   };

5683   return rewrite;
5684 }


5687 ////////////////////////////////////////////////////////////////////////////////
5688 //
5689 // BlockExp::reduce
5690 //
5691 ////////////////////////////////////////////////////////////////////////////////

5693 helper BlockExp::reduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
5694   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
5695   self.body->forEach(statement) {
5696     rewrite += statement.reduce(env, bt);
5697   };
5698   return rewrite;
5699 }


5702 ////////////////////////////////////////////////////////////////////////////////
5703 //
5704 // BlockExp::mixReduce
5705 //
5706 ////////////////////////////////////////////////////////////////////////////////
```

```
5708 helper BlockExp::mixReduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
5709   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
5710   self.body->forEach(statement) {
5711     rewrite += statement.mixReduce(env, bt);
5712   };
5713   return rewrite;
5714 }


5717 ////////////////////////////////////////////////////////////////////////////
5718 //
5719 // IfExp::reduce
5720 //
5721 ////////////////////////////////////////////////////////////////////////////

5723 helper IfExp::reduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
5724   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
5725   var sideEffects := createCollectionWrapper(rewrite);
5726   if self.condition.eval(env, bt).oclAsType(Boolean) then {
5727     rewrite += self.thenExpression.reduce(env, bt);
5728   }
5729   else {
5730     if self.elseExpression <> null then {
5731       rewrite += self.elseExpression.reduce(env, bt);
5732     }
5733     endif;
5734   }
5735   endif;
5736   return rewrite;
5737 }


5740 ////////////////////////////////////////////////////////////////////////////
5741 //
5742 // IfExp::mixReduce
5743 //
5744 ////////////////////////////////////////////////////////////////////////////

5746 helper IfExp::mixReduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
5747   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
5748   var sideEffects := createCollectionWrapper(rewrite);
5749   if self.condition.eval(env, bt).oclAsType(Boolean) then {
5750     rewrite += self.thenExpression.mixReduce(env, bt);
5751   }
5752   else {
5753     rewrite += self.elseExpression.mixReduce(env, bt);
5754   }
5755   endif;
5756   return rewrite;
5757 }


5760 ////////////////////////////////////////////////////////////////////////////
5761 //
5762 // VariableInitExp::reduce
5763 //
5764 ////////////////////////////////////////////////////////////////////////////

5766 helper VariableInitExp::reduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
5767   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
```

```
5768   var sideEffects := createCollectionWrapper(rewrite);
5769   var rvalue := self.referredVariable.initExpression.sideEffectsEval(env, bt, sideEffects);
5770   sideEffects.collection()->forEach(e) {
5771     rewrite += e.oclAsType(OCLExpression);
5772   };
5773   var newInitExp := self.deepclone().oclAsType(VariableInitExp);
5774   switch {
5775     case (rvalue.oclIsKindOf(EObject))
5776     {
5777       newInitExp.referredVariable.initExpression :=
5778         rvalue.oclAsType(EObject).makeExp()
5779       ;

5781     }
5782     else
5783     {
5784       newInitExp.referredVariable.initExpression := rvalue.makeExp();
5785     }
5786   };

5788   rewrite += newInitExp;

5790   getEnvironment(env).put(self.referredVariable.name, createFrame(rvalue));
5791   return rewrite;
5792 }


5795 ////////////////////////////////////////////////////////////////////////////
5796 //
5797 // ReturnExp::mixReduce
5798 //
5799 ////////////////////////////////////////////////////////////////////////////

5801 helper ReturnExp::mixReduce(env : OclAny ,bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
5802   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
5803   var valMix := self.value.mixReduce(env, bt);
5804   var newRet := self.deepclone().oclAsType(ReturnExp);
5805   newRet.value := valMix->last();
5806   if (valMix->size() > 1) then {
5807     rewrite += valMix->subOrderedSet(1, valMix->size() - 1);
5808   }
5809   endif;
5810   rewrite += newRet;
5811   return rewrite;
5812 }


5815 ////////////////////////////////////////////////////////////////////////////
5816 //
5817 // ASTNode::mixReduce
5818 //
5819 ////////////////////////////////////////////////////////////////////////////

5821 helper ASTNode::mixReduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
5822   return null;
5823 }


5826 ////////////////////////////////////////////////////////////////////////////
5827 //
5828 //  OCLExpression::mixReduce
```

```
5829 //
5830 ////////////////////////////////////////////////////////////////////////////////

5832 helper OCLExpression::mixReduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
5833   return OrderedSet{self};
5834 }


5837 ////////////////////////////////////////////////////////////////////////////////
5838 //
5839 // ASTNode::polyMixReduce
5840 //
5841 ////////////////////////////////////////////////////////////////////////////////

5843 helper ASTNode::polyMixReduce(env : OclAny, bt : Dict(String, BTAKind), context : EOperation) : OrderedSet(OCLExpression)
         {
5844   return null;
5845 }

5847 ////////////////////////////////////////////////////////////////////////////////
5848 //
5849 // OCLExpression::polyMixReduce
5850 //
5851 ////////////////////////////////////////////////////////////////////////////////

5853 helper OCLExpression::polyMixReduce(env : OclAny, bt : Dict(String, BTAKind), context : EOperation) : OrderedSet(
       OCLExpression) {
5854   return self.mixReduce(env, bt);
5855 }

5857 ////////////////////////////////////////////////////////////////////////////////
5858 //
5859 // VariableInitExp::mixReduce
5860 //
5861 ////////////////////////////////////////////////////////////////////////////////

5863 helper VariableInitExp::mixReduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
5864   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
5865   var newVarInit := self.deepclone().oclAsType(VariableInitExp);
5866 //  var staticEvalRewrite := self.reduce(env, bt);
5867   var staticResult := self.eval(env, bt);
5868   var mixed := self.referredVariable.initExpression.mixReduce(env, bt);
5869   var mixInit := mixed->last();
5870   var simplified := mixInit.toArithmaticExp().simplify().toOperationCall();
5871   var mergeExp := simplified.merge(staticResult, env, bt);
5872   var newInitExp : OCLExpression := mergeExp->last();
5873   if mixed->size() > 1 then {
5874     rewrite += mixed->subOrderedSet(1, mixed->size() - 1);
5875   }
5876   endif;
5877   if mergeExp->size() >1 then {
5878     rewrite += mergeExp->subOrderedSet(1, mergeExp->size() - 1);
5879   }
5880   endif;

5882   if newInitExp <> simplified then {
5883     var subVar := object VariableInitExp {
5884       referredVariable := object Variable {
5885         name := self.subName();
5886         initExpression := newInitExp.oclAsType(OperationCallExp).source;
5887       };
```

```
5888        name := referredVariable.name;
5889      };

5891      bt->put(subVar.referredVariable.name, BTAKind::DYNAMIC);
5892      subVar.eval(env, bt);
5893      rewrite += subVar;
5894      var initWithSubVar := newInitExp.oclAsType(OperationCallExp);
5895      initWithSubVar.source := object VariableExp {
5896        referredVariable := subVar.referredVariable.deepclone().oclAsType(Variable);
5897        name := referredVariable.oclAsType(Variable).name;
5898        eType := referredVariable.oclAsType(Variable).initExpression.getType().oclAsType(EClassifier);
5899      };
5900      newVarInit.referredVariable.initExpression := initWithSubVar.toArithmaticExp().simplify().toOperationCall();
5901    }
5902    else {
5903      newVarInit.referredVariable.initExpression := newInitExp;
5904    }
5905    endif;

5907    rewrite += newVarInit;
5908    bt->put(self.referredVariable.name, BTAKind::DYNAMIC);

5910    return rewrite;
5911  }


5914  ////////////////////////////////////////////////////////////////////////////
5915  //
5916  // VariableInitExp::polyMixReduce
5917  //
5918  ////////////////////////////////////////////////////////////////////////////

5920  helper VariableInitExp::polyMixReduce(env : OclAny, bt : Dict(String, BTAKind), context : EOperation) : OrderedSet(
        OCLExpression) {
5921    var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
5922    var newVarInit := self.deepclone().oclAsType(VariableInitExp);
5923  //  var staticEvalRewrite := self.reduce(env, bt);
5924    var staticResult := self.eval(env, bt);
5925    var staticResultName := '__' + context.name + self.referredVariable.fullName();
5926    var staticResultCache := staticResultName + 'Cache';
5927    var selfPath := getEnvironment(env).get('self').value().oclAsType(EObject).path();
5928    memoizeVariable(staticResultCache, selfPath, staticResult);
5929    var staticResultLookupExp := makeLookupExp(staticResultCache);
5930    var cacheType := getCacheType(staticResultCache);
5931    if cacheType.indexOf('ObjectPath') > 0 then {
5932      staticResultLookupExp := object OperationCallExp {
5933        source := object OperationCallExp {
5934          source := staticResultLookupExp.deepclone().oclAsType(OCLExpression);
5935          referredOperation := object EOperation {
5936            name := 'getObject';
5937          }.oclAsType(EObject);
5938          argument := OrderedSet{};
5939          argument += object VariableExp {
5940            name := this.context.inModel.name;
5941            referredVariable := object Variable {
5942              name := this.context.inModel.name;
5943            };
5944          };
5945          eType := staticResultLookupExp.eType.deepclone().oclAsType(EClassifier);
5946        };
5947        referredOperation := object EOperation {
```

```
5948          name := 'oclAsType';
5949        }.oclAsType(EObject);
5950        eType := staticResultLookupExp.eType.deepclone().oclAsType(EClassifier);
5951        argument := OrderedSet{};
5952        argument += object TypeExp {
5953          referredType := self.referredVariable.eType.getElementType();
5954        };
5955      };
5956    }
5957    endif;
5958    var mixed := self.referredVariable.initExpression.polyMixReduce(env, bt, context);
5959    var mixedInit := mixed->last();
5960    var simplified := mixedInit.toArithmaticExp().simplify().toOperationCall().oclAsType(ocl::ecore::OCLExpression);

5962    var mergeExp := simplified.merge(staticResultLookupExp, env, bt);
5963    var newInitExp : OCLExpression := mergeExp->last();
5964    if mixed->size() > 1 then {
5965      rewrite += mixed->subOrderedSet(1, mixed->size() - 1);
5966    }
5967    endif;
5968    if mergeExp->size() >1 then {
5969      rewrite += mergeExp->subOrderedSet(1, mergeExp->size() - 1);
5970    }
5971    endif;

5973    if newInitExp <> simplified then {
5974      var leftVar := self.referredVariable.deepclone().oclAsType(Variable);
5975      var subVarName := leftVar.subName();
5976      var subVarInit := newInitExp.oclAsType(OperationCallExp).source.oclAsType(OCLExpression);
5977      var subVar : OCLExpression;
5978      var initWithSubVar := newInitExp.oclAsType(OperationCallExp);
5979      if getEnvironment(env).hasKey(subVarName) then {
5980        var subLeftVar := object Variable {
5981          name := subVarName;
5982          initExpression := subVarInit;
5983        };
5984        subVar := object AssignExp {
5985          left := object VariableExp {
5986            name := subVarName;
5987            referredVariable := subLeftVar;
5988          };
5989          value := OrderedSet{subVarInit};
5990        };
5991        initWithSubVar.source := object VariableExp {
5992          referredVariable := subLeftVar;
5993          name := subVarName;
5994          eType := referredVariable.oclAsType(Variable).initExpression.getType().oclAsType(EClassifier);
5995        };
5996      }
5997      else {
5998        var subRefVar := object Variable {
5999          name := subVarName;
6000          initExpression := subVarInit;
6001        };
6002        subVar := object VariableInitExp {
6003          referredVariable := subRefVar;
6004          name := subVarName;
6005        };
6006        initWithSubVar.source := object VariableExp {
6007          referredVariable := subRefVar;
6008          name := subVarName;
```

```
6009          eType := referredVariable.oclAsType(Variable).initExpression.getType().oclAsType(EClassifier);
6010        };
6011      }
6012      endif;


6015      bt->put(subVar.name, BTAKind::DYNAMIC);
6016      subVar.eval(env, bt);
6017      rewrite += subVar;
6018      newVarInit.referredVariable.initExpression := initWithSubVar.toArithmaticExp().simplify().toOperationCall();
6019    }
6020    else {
6021      newVarInit.referredVariable.initExpression := newInitExp;
6022    }
6023    endif;
6024    newVarInit.referredVariable.eType := newVarInit.referredVariable.initExpression.getType().oclAsType(EClassifier);
6025    rewrite += newVarInit;
6026    bt->put(self.referredVariable.name, BTAKind::DYNAMIC);

6028    return rewrite;
6029  }

6031  ////////////////////////////////////////////////////////////////////////////
6032  //
6033  // ASTNode::fullName
6034  //
6035  ////////////////////////////////////////////////////////////////////////////

6037  helper ASTNode::fullName() : String {
6038    return null;
6039  }

6041  ////////////////////////////////////////////////////////////////////////////
6042  //
6043  // OCLExpresion::fullName
6044  //
6045  ////////////////////////////////////////////////////////////////////////////

6047  helper OCLExpression::fullName() : String {
6048    return self.name;
6049  }

6051  ////////////////////////////////////////////////////////////////////////////
6052  //
6053  // PropertyCallExp::fullName
6054  //
6055  ////////////////////////////////////////////////////////////////////////////

6057  helper PropertyCallExp::fullName() : String {
6058    var prop := self.referredProperty.oclAsType(EStructuralFeature).name;
6059    return '__' + self.source.getName() + '_' + prop;
6060  }

6062  ////////////////////////////////////////////////////////////////////////////
6063  //
6064  // VariableExp::fullName
6065  //
6066  ////////////////////////////////////////////////////////////////////////////

6068  helper VariableExp::fullName() : String {
6069    return self.referredVariable.fullName();
```

```
6070 }
6071
6072 ////////////////////////////////////////////////////////////////////////////////
6073 //
6074 // Variable::fullName
6075 //
6076 ////////////////////////////////////////////////////////////////////////////////
6077
6078 helper Variable::fullName() : String {
6079   return '__' + self.name;
6080 }
6081
6082 ////////////////////////////////////////////////////////////////////////////////
6083 //
6084 // ASTNode::subName
6085 //
6086 ////////////////////////////////////////////////////////////////////////////////
6087
6088 helper ASTNode::subName() : String {
6089   return self.fullName() + 'Sub';
6090 }
6091
6092 ////////////////////////////////////////////////////////////////////////////////
6093 //
6094 // OCLExpression::subName
6095 //
6096 ////////////////////////////////////////////////////////////////////////////////
6097
6098 helper OCLExpression::subName() : String {
6099   return self.fullName() + 'Sub';
6100 }
6101
6102
6103 ////////////////////////////////////////////////////////////////////////////////
6104 //
6105 // AssignExp::mixReduce
6106 //
6107 ////////////////////////////////////////////////////////////////////////////////
6108
6109 helper AssignExp::mixReduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
6110   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
6111   var newAssign := self.deepclone().oclAsType(AssignExp);
6112
6113 //  var staticEvalRewrite := self.reduce(env, bt);
6114   var staticResult := self.eval(env, bt);
6115   var mixed : OrderedSet(OCLExpression) := self.value->first().mixReduce(env, bt);
6116   var mixedValue := mixed->last();
6117   var simplified := mixedValue.toArithmaticExp().simplify().toOperationCall().oclAsType(ocl::ecore::OCLExpression);
6118   var mergeExp := simplified.merge(staticResult, env, bt);
6119   var newValue := mergeExp->last();
6120   //newAssign.value := OrderedSet{simplified};
6121   if mixed->size() > 1 then {
6122     rewrite += mixed->subOrderedSet(1, mixed->size() - 1);
6123   }
6124   endif;
6125   if mergeExp->size() > 1 then {
6126     rewrite += mergeExp->subOrderedSet(1, mergeExp->size() - 1);
6127   }
6128   endif;
6129   if newValue <> simplified then {
6130     var leftVar := self.left.deepclone().oclAsType(OCLExpression).reduce(env, bt)->last();
```

```
6131      var subVarName := leftVar.subName();
6132      var subVarValue := newValue.oclAsType(OperationCallExp).source.oclAsType(OCLExpression);
6133      var subVar : OCLExpression;
6134      var valueWithSubVar := newValue.oclAsType(OperationCallExp);
6135      if getEnvironment(env).hasKey(subVarName) then {
6136        var subLeftVar := object VariableExp {
6137          name := subVarName;
6138          referredVariable := object Variable {
6139            name := subVarName;
6140          };
6141        };
6142        subVar := object AssignExp {
6143          left := subLeftVar;
6144          value := OrderedSet{subVarValue};
6145        };
6146        valueWithSubVar.source := object VariableExp {
6147          referredVariable := subLeftVar.referredVariable;
6148          name := referredVariable.oclAsType(Variable).name;
6149          eType := new CollectionType();
6150        };
6151      }
6152      else {
6153        var subRefVar := object Variable {
6154          name := subVarName;
6155          initExpression := subVarValue;
6156        };
6157        subVar := object VariableInitExp {
6158          referredVariable := subRefVar;
6159          name := subVarName;
6160        };
6161        valueWithSubVar.source := object VariableExp {
6162          referredVariable := subRefVar;
6163          name := subVarName;
6164          eType := new CollectionType();
6165        };
6166      }
6167      endif;

6169      bt->put(subVar.name, BTAKind::DYNAMIC);
6170      subVar.eval(env, bt);
6171      rewrite += subVar;
6172      newAssign.value := OrderedSet{valueWithSubVar.toArithmaticExp().simplify().toOperationCall().oclAsType(OCLExpression)
            };
6173    }
6174    else {
6175      newAssign.value := OrderedSet{simplified};
6176    }
6177    endif;
6178    rewrite += newAssign;
6179    return rewrite;
6180  }

6182  ////////////////////////////////////////////////////////////////////////////
6183  //
6184  // AssignExp::polyMixReduce
6185  //
6186  ////////////////////////////////////////////////////////////////////////////

6188  helper AssignExp::polyMixReduce(env : OclAny, bt : Dict(String, BTAKind), context : EOperation) : OrderedSet(
          OCLExpression) {
6189    var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
```

```
6190   var newAssign := self.deepclone().oclAsType(AssignExp);

6192 //  var staticEvalRewrite := self.reduce(env, bt);
6193   var staticResult := self.eval(env, bt);
6194   var staticResultName := '__' + context.name + self.left.fullName();
6195   var staticResultCache := staticResultName + 'Cache';
6196   var selfPath := getEnvironment(env).get('self').value().oclAsType(EObject).path();
6197   memoizeVariable(staticResultCache, selfPath, staticResult);
6198   var staticResultLookupExp := makeLookupExp(staticResultCache);



6202   var mixed : OrderedSet(OCLExpression) := self.value->first().polyMixReduce(env, bt, context);
6203   var mixedValue := mixed->last();
6204   var simplified := mixedValue.toArithmaticExp().simplify().toOperationCall().oclAsType(ocl::ecore::OCLExpression);
6205   var mergeExp := simplified.merge(staticResultLookupExp, env, bt);
6206   var newValue := mergeExp->last();
6207   //newAssign.value := OrderedSet{simplified};
6208   if mixed->size() > 1 then {
6209     rewrite += mixed->subOrderedSet(1, mixed->size() - 1);
6210   }
6211   endif;
6212   if mergeExp->size() > 1 then {
6213     rewrite += mergeExp->subOrderedSet(1, mergeExp->size() - 1);
6214   }
6215   endif;
6216   if newValue <> simplified then {
6217     var leftVar := self.left.deepclone().oclAsType(OCLExpression).reduce(env, bt)->last();
6218     var subVarName := leftVar.subName();
6219     var subVarValue := newValue.oclAsType(OperationCallExp).source.oclAsType(OCLExpression);
6220     var subVar : OCLExpression;
6221     var valueWithSubVar := newValue.oclAsType(OperationCallExp);
6222     if getEnvironment(env).hasKey(subVarName) then {
6223       var subLeftVar := object VariableExp {
6224         name := subVarName;
6225         referredVariable := object Variable {
6226           name := subVarName;
6227         };
6228       };
6229       subVar := object AssignExp {
6230         left := subLeftVar;
6231         value := OrderedSet{subVarValue};
6232       };
6233       valueWithSubVar.source := object VariableExp {
6234         referredVariable := subLeftVar.referredVariable;
6235         name := referredVariable.oclAsType(Variable).name;
6236         eType := new CollectionType();
6237       };
6238     }
6239     else {
6240       var subRefVar := object Variable {
6241         name := subVarName;
6242         initExpression := subVarValue;
6243       };
6244       subVar := object VariableInitExp {
6245         referredVariable := subRefVar;
6246         name := subVarName;
6247       };
6248       valueWithSubVar.source := object VariableExp {
6249         referredVariable := subRefVar;
6250         name := subVarName;
```

```
6251         eType := new CollectionType();
6252       };
6253     }
6254     endif;

6256     bt->put(subVar.name, BTAKind::DYNAMIC);
6257     subVar.eval(env, bt);
6258     rewrite += subVar;
6259     newAssign.value := OrderedSet{valueWithSubVar.toArithmaticExp().simplify().toOperationCall().oclAsType(OCLExpression)
          };
6260   }
6261   else {
6262     newAssign.value := OrderedSet{simplified};
6263   }
6264   endif;
6265   rewrite += newAssign;
6266   return rewrite;
6267 }


6270 ////////////////////////////////////////////////////////////////////////////
6271 //
6272 // AssignExp::reduce
6273 //
6274 ////////////////////////////////////////////////////////////////////////////

6276 helper AssignExp::reduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
6277   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
6278   var rightBt := self.value->first().onlineBta(env, bt);
6279   var newAssignment := object AssignExp {
6280     left := self.left;
6281     value := self.value->first().eval(env, bt).makeExp();
6282   };
6283   rewrite += newAssignment;
6284   return rewrite;
6285 }

6287 ////////////////////////////////////////////////////////////////////////////
6288 //
6289 // ReturnExp::reduce
6290 //
6291 ////////////////////////////////////////////////////////////////////////////

6293 helper ReturnExp::reduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
6294   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
6295   var newRet := object ReturnExp {
6296     value := self.value.eval(env, bt).makeExp();
6297   };

6299   rewrite += newRet;
6300   return rewrite;
6301 }

6303 ////////////////////////////////////////////////////////////////////////////
6304 //
6305 // ConstructorBody::mixReduce
6306 //
6307 ////////////////////////////////////////////////////////////////////////////

6309 helper ConstructorBody::mixReduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
6310   var newBody : OrderedSet(OCLExpression) := OrderedSet{};
```

```
6311   self.content->forEach(expr) {
6312     newBody += expr.mixReduce(env, bt);
6313   };
6314   return newBody;
6315 }

6317 ////////////////////////////////////////////////////////////////////////////
6318 //
6319 // ConstructorBody::polyMixReduce
6320 //
6321 ////////////////////////////////////////////////////////////////////////////

6323 helper ConstructorBody::polyMixReduce(env : OclAny, bt : Dict(String, BTAKind), context : EOperation) : OrderedSet(
        OCLExpression) {
6324   var newBody : OrderedSet(OCLExpression) := OrderedSet{};
6325   self.content->forEach(expr) {
6326     newBody += expr.polyMixReduce(env, bt, context);
6327   };
6328   return newBody;
6329 }

6331 ////////////////////////////////////////////////////////////////////////////
6332 //
6333 // ObjectExp::mixReduce
6334 //
6335 ////////////////////////////////////////////////////////////////////////////

6337 helper ObjectExp::mixReduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
6338   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
6339   var obj := self.referredObject.name;
6340   var newBt : Dict(String, BTAKind) := Dict{};
6341   bt->keys()->forEach(k) {
6342     newBt->put(k, bt->get(k));
6343   };
6344   newBt->put(obj, BTAKind::DYNAMIC);
6345   var newObjectExp := self.deepclone().oclAsType(ObjectExp);
6346   newObjectExp.body := object ConstructorBody {
6347     content := self.body.mixReduce(env, newBt);
6348   };
6349   rewrite += newObjectExp;
6350   return rewrite;
6351 }

6353 ////////////////////////////////////////////////////////////////////////////
6354 //
6355 // ObjectExp::polyMixReduce
6356 //
6357 ////////////////////////////////////////////////////////////////////////////

6359 helper ObjectExp::polyMixReduce(env : OclAny, bt : Dict(String, BTAKind), context : EOperation) : OrderedSet(
        OCLExpression) {
6360   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
6361   var obj := self.referredObject.name;
6362   var newBt : Dict(String, BTAKind) := Dict{};
6363   bt->keys()->forEach(k) {
6364     newBt->put(k, bt->get(k));
6365   };
6366   newBt->put(obj, BTAKind::DYNAMIC);
6367   var newObjectExp := self.deepclone().oclAsType(ObjectExp);
6368   newObjectExp.body := object ConstructorBody {
6369     content := self.body.polyMixReduce(env, newBt, context);
```

```
6370   };
6371   rewrite += newObjectExp;
6372   return rewrite;
6373 }

6375 ////////////////////////////////////////////////////////////////////////////
6376 //
6377 // LogExp::mixReduce
6378 //
6379 ////////////////////////////////////////////////////////////////////////////

6381 helper LogExp::mixReduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
6382   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
6383   var args : OrderedSet(ocl::expressions::OCLExpression) := OrderedSet{};
6384   self.argument->forEach(a) {
6385     var argBt := a.onlineBta(env, bt);
6386     if (argBt = BTAKind::STATIC) then {
6387       args += a.eval(env, bt).makeExp();
6388     }
6389     else {
6390       args += a;
6391     }
6392     endif;
6393   };
6394   var newLog := object LogExp {
6395     argument := args;
6396   };

6398   rewrite += newLog;
6399   return rewrite;
6400 }

6402 ////////////////////////////////////////////////////////////////////////////
6403 //
6404 // MappingCallExp::mixReduce
6405 //
6406 ////////////////////////////////////////////////////////////////////////////

6408 helper MappingCallExp::mixReduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
6409   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
6410   var srcBT : BTAKind := self.source.onlineBta(env, bt);
6411   var argsBT := self.argument->onlineBta(env, bt);
6412   if argsBT->forAll(t|t = BTAKind::DYNAMIC) and srcBT = BTAKind::DYNAMIC then {
6413     var srcRedex := self.source.mixReduce(env, bt);

6415     var newCall := self.deepclone().oclAsType(MappingCallExp);

6417     newCall.source := srcRedex->last().deepclone().oclAsType(OCLExpression);

6419     var staticResult := self.eval(env, bt);
6420     var mergeExp := newCall.merge(staticResult, env, bt);
6421     var s := srcRedex->excluding(srcRedex->last())->asOrderedSet();
6422     s += mergeExp;
6423     rewrite += s;

6425     var newBt : Dict(String, BTAKind) := Dict{};
6426     bt->keys()->forEach(k) {
6427       newBt->put(k, bt->get(k));
6428     };

6430     var params := self.referredOperation.oclAsType(MappingOperation).eParameters->asOrderedSet();
```

```
6431      var callArgs := self.argument;

6433      var funcEnv := createEnvironment();
6434      funcEnv.parentEnv := getEnvironment(env);



6438      newBt->put('self', srcBT);
6439      var srcVal := self.source.eval(env, bt);
6440      funcEnv.put('self', createFrame(srcVal));
6441      if (srcBT = BTAKind::STATIC) then {
6442        funcEnv.put('self', createFrame(srcVal));
6443        newCall.source := srcVal.makeExp();
6444      }
6445      endif;


6448      var newCallArgs : OrderedSet(ocl::expressions::OCLExpression) := OrderedSet{};
6449      var newParams : OrderedSet(EParameter) := OrderedSet{};

6451      var i := 1;
6452      var n := params->size();
6453      while (i <= n) {
6454        var arg := callArgs->at(i);
6455        var param := params->at(i);
6456        var pbt := arg.onlineBta(env, bt);

6458        newBt->put(param.name, pbt);
6459        if (pbt = BTAKind::STATIC) then {
6460          var val := arg.eval(env, bt);
6461          funcEnv.put(param.name, createFrame(val));
6462          if (not self.referredOperation.oclIsTypeOf(MappingOperation)) then {
6463            newCallArgs += val.makeExp();
6464          }
6465          endif;
6466        }
6467        else {
6468          newCallArgs += arg.deepclone().oclAsType(OCLExpression);
6469          newParams += param.deepclone().oclAsType(EParameter);
6470        }
6471        endif;
6472        i := i + 1;
6473      };
6474      newCall.argument := newCallArgs;
6475      var isFirstMix := not mixCount->hasKey(self.referredOperation.oclAsType(MappingOperation).name);
6476      var mixedOp := self.referredOperation.oclAsType(MappingOperation).mix(funcEnv, newBt);
6477      mixedOp.eParameters := newParams;
6478      newCall.referredOperation := mixedOp.oclAsType(EObject);
6479      if isFirstMix then {
6480        newOperations += mixedOp;
6481      }
6482      else {
6483  //      var refOp := newCall.referredOperation.oclAsType(MappingOperation);
6484  //      refOp.name := refOp.name + '__mix1';
6485  //      newCall.referredOperation := refOp.oclAsType(EObject);
6486      }
6487      endif;

6489      rewrite += newCall;
6490    }
6491    endif;
```

```
6492   return rewrite;
6493 }

6495 ////////////////////////////////////////////////////////////////////////////
6496 //
6497 // OperationCallExp::mixReduce
6498 //
6499 ////////////////////////////////////////////////////////////////////////////

6501 helper OperationCallExp::mixReduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
6502   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
6503   var srcBT : BTAKind;
6504   var hasSource := self.source <> null or not self.source.oclIsInvalid();
6505   if hasSource then {
6506     srcBT := self.source.onlineBta(env, bt);
6507   }
6508   else {
6509     srcBT := BTAKind::STATIC;
6510   }
6511   endif;
6512   var argsBT := self.argument->onlineBta(env, bt);
6513   if (argsBT->forAll(b|b = BTAKind::DYNAMIC) and (not hasSource or srcBT = BTAKind::DYNAMIC)) then {
6514     //return self.reduce(env, bt);
6515     var srcRedex := self.source.mixReduce(env, bt);
6516     var newCall := self.deepclone().oclAsType(OperationCallExp);
6517     newCall.source := srcRedex->last().deepclone().oclAsType(OCLExpression);
6518 //    var staticResult := self.eval(env, bt);
6519 //    var mergeExp := newCall.merge(staticResult);
6520     if srcRedex->size() > 1 then {
6521       rewrite += srcRedex->subOrderedSet(1, srcRedex->size() - 1);
6522     }
6523     endif;
6524 //    var s := srcRedex;//->excluding(srcRedex->last())->asOrderedSet();
6525 //    s += mergeExp;
6526 //    return s;
6527     rewrite += newCall;
6528     return rewrite;
6529   }
6530   endif;
6531   var newBt : Dict(String, BTAKind) := Dict{};
6532   bt->keys()->forEach(k) {
6533     newBt->put(k, bt->get(k));
6534   };

6536   var params := self.referredOperation.oclAsType(EOperation).eParameters->asOrderedSet();
6537   var callArgs := self.argument;

6539   var funcEnv := createEnvironment();
6540   funcEnv.parentEnv := getEnvironment(env);
6541   var newCall := self.deepclone().oclAsType(OperationCallExp);

6543   if (hasSource) then {
6544     newBt->put('self', srcBT);

6546     if (srcBT = BTAKind::STATIC) then {
6547       var srcVal := self.source.eval(env, bt);
6548       funcEnv.put('self', createFrame(srcVal));
6549       newCall.source := srcVal.makeExp();
6550     }
6551     else {
6552       var srcMix := self.source.mixReduce(env, bt);
```

```
6553        if srcMix->size() > 1 then {
6554          rewrite += srcMix->subOrderedSet(1, srcMix->size() - 1);
6555        }
6556        endif;
6557        newCall.source := srcMix->last();
6558      }
6559      endif;
6560    }
6561    endif;

6563    var newCallArgs : OrderedSet(ocl::expressions::OCLExpression) := OrderedSet{};
6564    var newParams : OrderedSet(EParameter) := OrderedSet{};

6566    var i := 1;
6567    var n := params->size();
6568    while (i <= n) {
6569      var arg := callArgs->at(i);
6570      var param := params->at(i);
6571      var pbt := arg.onlineBta(env, bt);

6573      newBt->put(param.name, pbt);
6574      if (pbt = BTAKind::STATIC) then {
6575        var val := arg.eval(env, bt);
6576        funcEnv.put(param.name, createFrame(val));
6577        if (not self.referredOperation.oclIsTypeOf(Helper)) then {
6578          newCallArgs += val.makeExp();
6579        }
6580        endif;
6581      }
6582      else {
6583        var argMix := arg.mixReduce(env, bt);
6584        if argMix->size() > 1 then {
6585          rewrite += argMix->subOrderedSet(1, argMix->size() - 1);
6586        }
6587        endif;
6588        newCallArgs += argMix->last(); //arg.deepclone().oclAsType(OCLExpression);
6589        newParams += param.deepclone().oclAsType(EParameter);
6590      }
6591      endif;
6592      i := i + 1;
6593    };
6594    newCall.argument := newCallArgs;

6596    if (not self.referredOperation.oclIsTypeOf(Helper)) then {
6597      rewrite += newCall;
6598      return rewrite;
6599    }
6600    endif;
6601    var mixedOp := self.referredOperation.oclAsType(Helper).mix(funcEnv, newBt);
6602    mixedOp.eParameters := newParams;

6604    newOperations += mixedOp;

6606    newCall.referredOperation := mixedOp.oclAsType(EObject);

6608    rewrite += newCall;

6610    return rewrite;
6611  }

6613  ////////////////////////////////////////////////////////////////////////////
```

```
6614 //
6615 // OperationCallExp::polyMixReduce
6616 //
6617 ////////////////////////////////////////////////////////////////////////////

6619 helper OperationCallExp::polyMixReduce(env : OclAny, bt : Dict(String, BTAKind), context : EOperation) : OrderedSet(
         OCLExpression) {
6620   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
6621   var srcBT : BTAKind;
6622   var hasSource := self.source <> null or not self.source.oclIsInvalid();
6623   if hasSource then {
6624     srcBT := self.source.onlineBta(env, bt);
6625   }
6626   else {
6627     srcBT := BTAKind::STATIC;
6628   }
6629   endif;
6630   var argsBT := self.argument->onlineBta(env, bt);
6631   if (argsBT->forAll(b|b = BTAKind::DYNAMIC) and (not hasSource or srcBT = BTAKind::DYNAMIC)) then {
6632     //return self.reduce(env, bt);
6633     var srcRedex := self.source.polyMixReduce(env, bt, context);
6634     var newCall := self.deepclone().oclAsType(OperationCallExp);
6635     newCall.source := srcRedex->last().deepclone().oclAsType(OCLExpression);
6636 //    var staticResult := self.eval(env, bt);
6637 //    var mergeExp := newCall.merge(staticResult);
6638     if srcRedex->size() > 1 then {
6639       rewrite += srcRedex->subOrderedSet(1, srcRedex->size() - 1);
6640     }
6641     endif;
6642 //    var s := srcRedex;//->excluding(srcRedex->last())->asOrderedSet();
6643 //    s += mergeExp;
6644 //    return s;
6645     rewrite += newCall;
6646     return rewrite;
6647   }
6648   endif;
6649   var newBt : Dict(String, BTAKind) := Dict{};
6650   bt->keys()->forEach(k) {
6651     newBt->put(k, bt->get(k));
6652   };

6654   var params := self.referredOperation.oclAsType(EOperation).eParameters->asOrderedSet();
6655   var callArgs := self.argument;

6657   var funcEnv := createEnvironment();
6658   funcEnv.parentEnv := getEnvironment(env);
6659   var newCall := self.deepclone().oclAsType(OperationCallExp);

6661   if (hasSource) then {
6662     newBt->put('self', srcBT);

6664     if (srcBT = BTAKind::STATIC) then {
6665       var srcVal := self.source.eval(env, bt);
6666       funcEnv.put('self', createFrame(srcVal));
6667       newCall.source := srcVal.makeExp();
6668     }
6669     endif;
6670   }
6671   endif;

6673   var newCallArgs : OrderedSet(ocl::expressions::OCLExpression) := OrderedSet{};
```

```
6674   var newParams : OrderedSet(EParameter) := OrderedSet{};

6676   var i := 1;
6677   var n := params->size();
6678   while (i <= n) {
6679     var arg := callArgs->at(i);
6680     var param := params->at(i);
6681     var pbt := arg.onlineBta(env, bt);

6683     newBt->put(param.name, pbt);
6684     if (pbt = BTAKind::STATIC) then {
6685       var val := arg.eval(env, bt);
6686       funcEnv.put(param.name, createFrame(val));
6687       if (not self.referredOperation.oclIsTypeOf(Helper)) then {
6688         newCallArgs += val.makeExp();
6689       }
6690       endif;
6691     }
6692     else {
6693       newCallArgs += arg.deepclone().oclAsType(OCLExpression);
6694       newParams += param.deepclone().oclAsType(EParameter);
6695     }
6696     endif;
6697     i := i + 1;
6698   };
6699   newCall.argument := newCallArgs;

6701   if (not self.referredOperation.oclIsTypeOf(Helper)) then {
6702     rewrite += newCall;
6703     return rewrite;
6704   }
6705   endif;
6706   var mixedOp := self.referredOperation.oclAsType(Helper).mix(funcEnv, newBt);
6707   mixedOp.eParameters := newParams;

6709   newOperations += mixedOp;

6711   newCall.referredOperation := mixedOp.oclAsType(EObject);

6713   rewrite += newCall;

6715   return rewrite;
6716 }

6718 ////////////////////////////////////////////////////////////////////////////
6719 //
6720 // ImperativeIterateExp::mixReduce
6721 //
6722 ////////////////////////////////////////////////////////////////////////////

6724 helper ImperativeIterateExp::mixReduce(env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
6725   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
6726   var srcVariable : VariableInitExp;
6727   if self.source.oclAsType(ETypedElement).eType.oclIsKindOf(CollectionType) then {
6728     var srcMix := self.source.mixReduce(env, bt);
6729     if srcMix->size() > 1 then {
6730       rewrite += srcMix->subOrderedSet(1, srcMix->size() - 1);
6731     }
6732     endif;
6733     srcVariable := object VariableInitExp {
6734       referredVariable := object Variable {
```

```
6735          name := newTemp();
6736          initExpression := srcMix->last().deepclone().oclAsType(OCLExpression);
6737        };
6738        name := referredVariable.name;
6739      };
6740    }
6741    else {
6742      srcVariable := self.makeSourceVariable(newTemp());
6743    }
6744    endif;
6745    var environ := getEnvironment(env);

6747    bt->put(srcVariable.name, BTAKind::DYNAMIC);
6748    var srcVal := srcVariable.eval(env, bt);
6749    rewrite += srcVariable;

6751    var last := getCollectionWrapper(environ.get(srcVariable.referredVariable.name).value()).collection()->size();
6752    var lastVarName := srcVariable.referredVariable.name + 'Last';
6753    bt->put(lastVarName, BTAKind::DYNAMIC);
6754    var lastVar := makeVar(lastVarName, last);
6755    var lastVal := lastVar.eval(env, bt);
6756    rewrite += lastVar;

6758    var sizeVar := makeSizeVar(srcVariable.referredVariable);
6759    bt->put(sizeVar.name, BTAKind::DYNAMIC);
6760    var sizeVal := sizeVar.eval(environ, bt);
6761    rewrite += sizeVar;

6763    var subSetVar := makeSubSetVar(srcVariable.referredVariable);
6764    bt->put(subSetVar.name, BTAKind::DYNAMIC);
6765    var subSetVal := subSetVar.eval(env, bt);
6766    rewrite += subSetVar;

6768    var subSetVarExp := object VariableExp {
6769      referredVariable := subSetVar.referredVariable.deepclone().oclAsType(Variable);
6770      name := referredVariable.oclAsType(Variable).name;
6771    };

6773    var imperativeExp := self.deepclone().oclAsType(ImperativeIterateExp);
6774    imperativeExp.source := subSetVarExp;
6775    if self.name = 'xcollect' then {
6776      var newBt : Dict(String, BTAKind) = Dict{};
6777      bt->keys()->forEach(k) {
6778        newBt->put(k, bt->get(k));
6779      };
6780      newBt->put(self.iterator->first().getName(), BTAKind::DYNAMIC);
6781      var newEnv := createEnvironment(getEnvironment(env));
6782      var newBody : OCLExpression;

6784      getCollectionWrapper(self.source.eval(env, newBt)).collection()->forEach(element) {
6785        newEnv.put(self.iterator->first().getName(), createFrame(element));
6786        var bodyMix := self.body.mixReduce(newEnv, newBt);
6787        if bodyMix->size() > 1 then {
6788          rewrite += bodyMix->subOrderedSet(1, bodyMix->size() - 1);
6789        }
6790        endif;
6791        newBody := bodyMix->last();
6792      };
6793      //imperativeExp.body := newBody;
6794      var union := object OperationCallExp {
6795        source := imperativeExp;
```

```
6796        referredOperation := object EOperation {
6797          name := 'union';
6798        }.oclAsType(EObject);
6799        argument := OrderedSet{};
6800        argument += object ImperativeIterateExp {
6801          name := self.name;
6802          source := object OperationCallExp {
6803            referredOperation := object EOperation {
6804              name := 'subOrderedSet';
6805            }.oclAsType(EObject);
6806            source := self.source.deepclone().oclAsType(OCLExpression);
6807            argument := OrderedSet{};
6808            argument += 1.makeExp();
6809            argument += object VariableExp {
6810              referredVariable := lastVar.referredVariable;
6811              name := lastVar.referredVariable.name;
6812            };
6813          };
6814          iterator := self.iterator;
6815          body := newBody;

6817        };
6818      };
6819      rewrite += union;
6820    }
6821    endif;
6822    return rewrite;
6823  }

6825  ////////////////////////////////////////////////////////////////////////////
6826  //
6827  // ImperativeIterateExp::polyMixReduce
6828  //
6829  ////////////////////////////////////////////////////////////////////////////

6831  helper ImperativeIterateExp::polyMixReduce(env : OclAny, bt : Dict(String, BTAKind), context : EOperation) : OrderedSet(
          OCLExpression) {
6832    var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
6833    var selfPath := getEnvironment(env).get('self').value().oclAsType(EObject).path();
6834    var srcVariable : VariableInitExp;
6835    var srcVarName := '__' + context.name + newTemp();
6836    if self.source.oclAsType(ETypedElement).eType.oclIsKindOf(CollectionType) then {
6837      var srcMix := self.source.polyMixReduce(env, bt, context);
6838      if srcMix->size() > 1 then {
6839        rewrite += srcMix->subOrderedSet(1, srcMix->size() - 1);
6840      }
6841      endif;
6842      srcVariable := object VariableInitExp {
6843        referredVariable := object Variable {
6844          name := srcVarName;
6845          initExpression := srcMix->last().deepclone().oclAsType(OCLExpression);
6846        };
6847        name := referredVariable.name;
6848      };
6849    }
6850    else {
6851      srcVariable := self.makeSourceVariable(srcVarName);
6852    }
6853    endif;
6854    var environ := getEnvironment(env);
```

```
6856    bt->put(srcVariable.name, BTAKind::DYNAMIC);
6857    var srcVal := srcVariable.eval(env, bt);
6858    rewrite += srcVariable;

6860    var last := getCollectionWrapper(environ.get(srcVariable.referredVariable.name).value()).collection()->size();
6861    var lastVarName := srcVarName + 'Last';
6862    var lastVarCache := lastVarName + 'Cache';
6863    bt->put(lastVarName, BTAKind::DYNAMIC);
6864    memoizeVariable(lastVarCache, selfPath, last);
6865    var lookupExp := makeLookupExp(lastVarCache);

6867    var lastVar := makeVar(lastVarName, lookupExp);
6868    //var lastVal := lastVar.eval(env, bt);
6869    getEnvironment(env).put(lastVarName, createFrame(last));
6870    rewrite += lastVar;

6872    var sizeVar := makeSizeVar(srcVariable.referredVariable);
6873    bt->put(sizeVar.name, BTAKind::DYNAMIC);
6874    var sizeVal := sizeVar.eval(environ, bt);
6875    rewrite += sizeVar;

6877    var subSetVar := makeSubSetVar(srcVariable.referredVariable);
6878    bt->put(subSetVar.name, BTAKind::DYNAMIC);
6879    var subSetVal := subSetVar.eval(env, bt);
6880    rewrite += subSetVar;

6882    var subSetVarExp := object VariableExp {
6883      referredVariable := subSetVar.referredVariable.deepclone().oclAsType(Variable);
6884      name := referredVariable.oclAsType(Variable).name;
6885    };

6887    var imperativeExp := self.deepclone().oclAsType(ImperativeIterateExp);
6888    imperativeExp.source := subSetVarExp;
6889    // Specialize Collect expression
6890    if self.name = 'xcollect' then {
6891      var newBt : Dict(String, BTAKind) := Dict{};
6892      bt->keys()->forEach(k) {
6893        newBt->put(k, bt->get(k));
6894      };
6895      newBt->put(self.iterator->first().getName(), BTAKind::DYNAMIC);
6896      var newEnv := createEnvironment(getEnvironment(env));
6897      var newBody : OCLExpression;
6898      getCollectionWrapper(self.source.eval(env, newBt)).collection()->forEach(element) {
6899        newEnv.put(self.iterator->first().getName(), createFrame(element));
6900        var bodyMix := self.body.mixReduce(newEnv, newBt);
6901        if bodyMix->size() > 1 then {
6902          rewrite += bodyMix->subOrderedSet(1, bodyMix->size() - 1);
6903        }
6904        endif;
6905        newBody := bodyMix->last();
6906      };
6907      imperativeExp.body := newBody;
6908    }
6909    endif;
6910    rewrite += imperativeExp;
6911    return rewrite;
6912  }


6915  ////////////////////////////////////////////////////////////////////////////
6916  //
```

```
6917 // ASTNode::merge
6918 //
6919 ////////////////////////////////////////////////////////////////////////////////


6922 helper ASTNode::merge(val : OclAny, env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
6923   return null;
6924 }

6926 ////////////////////////////////////////////////////////////////////////////////
6927 //
6928 // OCLExpression::merge
6929 //
6930 ////////////////////////////////////////////////////////////////////////////////

6932 helper OCLExpression::merge(val : OclAny, env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
6933   return OrderedSet{self};
6934 }

6936 ////////////////////////////////////////////////////////////////////////////////
6937 //
6938 // ASTNode::merge
6939 //
6940 ////////////////////////////////////////////////////////////////////////////////

6942 helper ASTNode::getSub(env : OclAny, bt : Dict(String, BTAKind)) : OCLExpression {
6943   return null;
6944 }

6946 ////////////////////////////////////////////////////////////////////////////////
6947 //
6948 // OCLExpression::merge
6949 //
6950 ////////////////////////////////////////////////////////////////////////////////

6952 helper OCLExpression::getSub(env : OclAny, bt : Dict(String, BTAKind)) : OCLExpression {
6953   return self;
6954 }

6956 ////////////////////////////////////////////////////////////////////////////////
6957 //
6958 // VariableExp::merge
6959 //
6960 ////////////////////////////////////////////////////////////////////////////////

6962 helper VariableExp::getSub(env : OclAny, bt : Dict(String, BTAKind)) : OCLExpression {
6963   var subName := self.subName();
6964   if getEnvironment(env).hasKey(subName) then {
6965     var newVar := object VariableExp {
6966       name := subName;
6967       referredVariable := object Variable {
6968         name := subName;
6969         eType := self.eType.deepclone().oclAsType(EClassifier);
6970       };
6971       eType := referredVariable.oclAsType(Variable).eType;
6972     };
6973     return newVar;
6974   }
6975   endif;
6976   return self;
6977 }
```

```
6979 ////////////////////////////////////////////////////////////////////////////
6980 //
6981 // OperationCallExp::merge
6982 //
6983 ////////////////////////////////////////////////////////////////////////////

6985 helper OperationCallExp::merge(val : OclAny, env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
6986   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
6987   if self.referredOperation.oclIsKindOf(Helper) then {
6988     return OrderedSet{self};
6989   }
6990   endif;
6991   var opName := self.referredOperation.oclAsType(EOperation).name;

6993   var subSrc := self.source.getSub(env, bt);

6995   var newCall : OperationCallExp;
6996   if subSrc <> self.source then {
6997     newCall := self.deepclone().oclAsType(OperationCallExp);
6998     newCall.source := subSrc;
6999   }
7000   else {
7001     newCall := self; // var y := 2 * xd should not be replaced by var y:= 2 * xs (the static value of xd), hende
          returning here.
7002     if self.source.onlineBta(env, bt) = BTAKind::STATIC then {
7003       rewrite += newCall;
7004       return rewrite;
7005     }
7006     endif;
7007   }
7008   endif;

7010   switch {
7011     case (opName = 'sum')
7012     {
7013       rewrite += mergeSum(newCall, val);
7014     }
7015     case (opName = '+' or opName = '*')
7016     {
7017       var argBt := newCall.argument->first().onlineBta(env, bt);
7018       if (argBt = BTAKind::STATIC) then {
7019         newCall.argument := OrderedSet{val.makeExp()};
7020       }
7021       endif;
7022       rewrite += newCall;
7023     }
7024     case (opName = '-')
7025     {
7026       newCall.argument := OrderedSet{val.makeExp()};
7027       newCall.referredOperation.oclAsType(EOperation).name := '+';
7028       rewrite += newCall;
7029     }
7030     case (opName = 'indexOf')
7031     {
7032       rewrite += val.makeExp();
7033     }
7034     else
7035     {
7036       rewrite += newCall;
7037     }
```

```
7038   };

7040   return rewrite;
7041 }

7043 ////////////////////////////////////////////////////////////////////////////
7044 //
7045 // Imperative::IterateExp::merge
7046 //
7047 ////////////////////////////////////////////////////////////////////////////

7049 helper ImperativeIterateExp::merge(val : OclAny, env : OclAny, bt : Dict(String, BTAKind)) : OrderedSet(OCLExpression) {
7050   var rewrite : OrderedSet(OCLExpression) := OrderedSet{};
7051   switch {
7052     case (self.name = 'xcollect' or self.name = 'xselect')
7053     {
7054       var union := object OperationCallExp {
7055         referredOperation := object EOperation {
7056           name := 'union';
7057         }.oclAsType(EObject);
7058         source := self.deepclone().oclAsType(OCLExpression);
7059         argument := OrderedSet{};
7060         argument += object OperationCallExp {
7061           referredOperation := object EOperation {
7062             name := 'as' +
7063               if self.eType.oclAsType(CollectionType).kind = ocl::expressions::CollectionKind::Set then 'Set' else 'Bag'
                      endif;
7064           }.oclAsType(EObject);
7065           source := val.makeExp();

7067         };
7068       };
7069       rewrite += union;
7070     }
7071     else
7072     {
7073       rewrite += self;
7074     }
7075   };
7076   return rewrite;
7077 }

7079 ////////////////////////////////////////////////////////////////////////////
7080 //
7081 // mergeHom
7082 //
7083 ////////////////////////////////////////////////////////////////////////////

7085 helper OperationCallExp::mergeHom(val : OclAny) : OrderedSet(OCLExpression) {
7086   var res : OrderedSet(OCLExpression) := OrderedSet{};
7087   var mergeExp := object OperationCallExp {
7088     source := self;
7089     argument := val.makeExp();
7090     referredOperation := self.referredOperation.deepclone().oclAsType(EObject);
7091   };
7092   res += mergeExp;
7093   return res;
7094 }

7096 ////////////////////////////////////////////////////////////////////////////
7097 //
```

```
7098 // mergeSum
7099 //
7100 ////////////////////////////////////////////////////////////////////////////

7102 helper mergeSum(sumOp : OperationCallExp, val : OclAny) : OrderedSet(OCLExpression) {
7103   var res : OrderedSet(OCLExpression) := OrderedSet{};
7104   var mergeExp := object OperationCallExp {
7105     source := sumOp;
7106     argument := val.makeExp();
7107     referredOperation := object EOperation {
7108       name := '+';
7109     }.oclAsType(EObject);
7110   };

7112   res += mergeExp;
7113   return res;
7114 }

7116 ////////////////////////////////////////////////////////////////////////////
7117 //
7118 // makeVar
7119 //
7120 ////////////////////////////////////////////////////////////////////////////


7123 helper makeVar(varName : String, val : OclAny) : VariableInitExp {
7124   return object VariableInitExp {
7125     referredVariable := object Variable {
7126       name := varName;
7127       initExpression := val.makeExp();
7128     };
7129     name := referredVariable.name;
7130   };
7131 }

7133 ////////////////////////////////////////////////////////////////////////////
7134 //
7135 // makeSizeVar
7136 //
7137 ////////////////////////////////////////////////////////////////////////////

7139 helper makeSizeVar(variable : Variable) : VariableInitExp {
7140   return object VariableInitExp {
7141     referredVariable := object Variable {
7142       name := variable.name + 'Size';
7143       initExpression := object OperationCallExp {
7144         referredOperation := object EOperation {
7145           name := 'size';
7146         }.oclAsType(EObject);
7147         source := object VariableExp {
7148           referredVariable := variable;
7149           name := referredVariable.oclAsType(Variable).name;
7150           eType := new CollectionType();
7151         };
7152       };
7153     };
7154     name := referredVariable.name;
7155   };
7156 }

7158 ////////////////////////////////////////////////////////////////////////////
```

```
7159 //
7160 // makeSubsetVar
7161 //
7162 ////////////////////////////////////////////////////////////////////////////

7164 helper makeSubSetVar(variable : Variable) : VariableInitExp {

7166   return object VariableInitExp {
7167     referredVariable := object Variable {
7168       name := variable.name + 'Sub';
7169       initExpression :=  object IfExp {
7170         condition := object OperationCallExp {
7171           referredOperation := object EOperation {
7172             name := '>';
7173           }.oclAsType(EObject);
7174           source := object VariableExp {
7175             referredVariable := object Variable {
7176               name := variable.name + 'Size';
7177             };
7178             name := referredVariable.oclAsType(Variable).name;
7179           };
7180           argument := OrderedSet{
7181             object VariableExp {
7182               referredVariable := object Variable {
7183                 name := variable.name + 'Last';
7184               };
7185               name := referredVariable.oclAsType(Variable).name;
7186             }
7187           }
7188         };
7189         thenExpression := object OperationCallExp {
7190           source := object VariableExp {
7191             referredVariable := variable;
7192             name := referredVariable.oclAsType(Variable).name;
7193             eType := new CollectionType();
7194           };
7195           referredOperation := object EOperation {
7196             name := 'subOrderedSet';
7197           }.oclAsType(EObject);
7198           argument += object OperationCallExp {
7199             referredOperation := object EOperation {
7200               name := '+';
7201             }.oclAsType(EObject);
7202             source := object VariableExp {
7203               referredVariable := object Variable {
7204                 name := variable.name + 'Last';
7205               };
7206               name := referredVariable.oclAsType(Variable).name;
7207             };
7208             argument += object IntegerLiteralExp {
7209               integerSymbol := 1;
7210             };
7211           };
7212           argument += object VariableExp {
7213             referredVariable := object Variable {
7214               name := variable.name + 'Size';
7215             };
7216             name := referredVariable.oclAsType(Variable).name;
7217           };
7218         };
7219         elseExpression := object CollectionLiteralExp {
```

```
7220          kind := ocl::expressions::CollectionKind::OrderedSet;
7221          eType := object OrderedSetType {
7222            elementType := variable.eType.oclAsType(EObject);
7223          };
7224        };
7225      };
7226    };
7227    name := referredVariable.name;
7228  };
7229 }


7232 ////////////////////////////////////////////////////////////////////////////
7233 //
7234 // makeSourceVariable
7235 //
7236 ////////////////////////////////////////////////////////////////////////////

7238 helper ImperativeIterateExp::makeSourceVariable(varName : String) : VariableInitExp {

7240   var initExp := self.source.deepclone().oclAsType(OCLExpression);
7241   if (self.source.oclIsKindOf(ETypedElement)) then {
7242     var type := self.source.oclAsType(ETypedElement).eType;
7243     if (type.oclIsKindOf(SetType)) then {
7244       initExp := object OperationCallExp {
7245         referredOperation := object EOperation {
7246           name := 'asOrderdSet';
7247         }.oclAsType(EObject);
7248         source := initExp;
7249       };
7250     }
7251     endif;
7252   }
7253   endif;

7255   return object VariableInitExp {
7256     referredVariable := object Variable {
7257       name := varName;
7258       initExpression := initExp;
7259     };
7260     name := referredVariable.name;
7261   };
7262 }

7264 ////////////////////////////////////////////////////////////////////////////
7265 //
7266 // makeLookupExp
7267 //
7268 ////////////////////////////////////////////////////////////////////////////

7270 helper makeLookupExp(cacheName : String) : OCLExpression {
7271   var lookupExp := object OperationCallExp {
7272     source := object VariableExp {
7273       name := cacheName;
7274       referredVariable := object Variable {
7275         name := cacheName;
7276       };
7277       eType := object DictionaryType {
7278         name := 'Dict(String, OclAny)';
7279       };
7280     };
```

```
7283      referredOperation := object EOperation {
7284        name := 'get';
7285        argument := OrderedSet{
7286          object OperationCallExp {
7287            source := object OperationCallExp {
7288              source := object VariableExp {
7289                name := 'self';
7290                referredVariable := object Variable {
7291                  name := 'self';
7292                };
7293                eType := object EClass {
7294                  name := 'EObject';
7295                };
7296              };


7299              referredOperation := object EOperation {
7300                name := 'oclAsType';
7301                argument := OrderedSet{
7302                  object TypeExp {
7303                    referredType := object EClass {
7304                      name := 'EObject';
7305                    }.oclAsType(EObject);
7306                  }
7307                };

7309                eType := object EClass {
7310                  name := 'EObject';
7311                };
7312              }.oclAsType(EObject);
7313            };

7315            referredOperation := object EOperation {
7316              name := 'path';
7317            }.oclAsType(EObject);
7318          }
7319        };
7320      }.oclAsType(EObject);

7322      eType := memoizeTables->get(cacheName).eType.getElementType().oclAsType(EClassifier);
7323    };

7325    return lookupExp;
7326  }
```

Appendix E

Change Factorization Algorithm Implementation in C++

C++ Code

```
1  //////////  Change Factorization
2  #include <iostream>
3  #include <sstream>
4  #include <string>
5  #include <vector>
6  #include <iterator>
7  #include <map>
8  #include <cstdlib>
9  #include <ctime>

11 enum ChangeType
12 {
13     NOP,
14     UPD,
15     DEL,
16     INS
17 };

19 int cost(ChangeType c)
20 {
21     switch (c) {
22     case NOP:
23         return 0;
24     case UPD:
25     case DEL:
26     case INS:
27         return 1;
28     }
29 }

31 std::string print(ChangeType c)
32 {
```

```
    switch (c) {
    case NOP:
        return "NOP";
    case UPD:
        return "UPD";
    case DEL:
        return "DEL";
    case INS:
        return "INS";
    }
}

struct Node
{
    Node()
    {}

    Node(const std::string& v) : label(v)
    {}

    std::string label;
    std::vector<Node*> children;
};

bool operator==(const Node& n1, const Node& n2)
{
    if (n1.label != n2.label) {
        return false;
    }
    return n1.children == n2.children;
}

std::string print(const Node& node)
{
    std::string s = node.label;
    std::vector<Node*>::const_iterator
            it = node.children.begin(),
            itend = node.children.end();
    if (it == itend)
        return s;
    s += "{" + print(**it);
    ++it;
    for (; it != itend; ++it) {
        s += "," + print(**it);
    }
    s+= "}";

    return s;
}

Node pruneLast(const Node& node)
{
    Node n = node;
    n.children.pop_back();
    return n;
}

Node last(const Node& node)
{
    if (node.children.empty())
        return Node();
```

```
    else
        return *(node.children.back());
}

typedef std::map<std::pair<std::string, std::string>, int> DistanceTable;
typedef std::map<std::pair<std::string, std::string>, ChangeType> ChangeTable;

DistanceTable dt;
ChangeTable ct;
int hits = 0;

int dist(const Node& s, const Node& t)
{
   std::pair<std::string, std::string> key(print(s),print(t));
   DistanceTable::iterator it = dt.find(key);

   if (it != dt.end()) {
        ++hits;
        return it->second;
   }
   if (t.children.empty() && s.children.empty()) {
        if (t.label.empty()) {
            dt[key] = 0;
            ct[key] = NOP;
            return 0;
        }
        if (t.label == s.label) {
            dt[key] = 0;
            ct[key] = NOP;
            return 0;
        }
        else {
            ct[key] = UPD;
            dt[key] = cost(UPD);
            return cost(UPD);
        }
   }

   if (s.children.empty()) {
        ct[key] = INS;
        int mc = cost(INS) + dist(s, pruneLast(t)) + dist(Node(), last(t));
        dt[key] = mc;
        return mc;
   }

   if (t.children.empty()) {
        int mc = cost(DEL);
        ct[key] = DEL;
        dt[key] = mc;
        return mc;
   }

   int updCost = dist(pruneLast(s), pruneLast(t)) + dist(last(s), last(t));
   int insCost = dist(s, pruneLast(t)) + dist(Node(), last(t)) + cost(INS);
   int delCost = dist(pruneLast(s), t) + cost(DEL);

   int minCost = updCost;
   ChangeType change = UPD;

   if (insCost < updCost) {
        if (insCost < delCost) {
```

```
            minCost = insCost;
            change = INS;
        }
        else {
            minCost = delCost;
            change = DEL;
        }
    }
    else if (delCost < updCost) {
         minCost = delCost;
         change = DEL;
    }
    ct[key] = change;
    dt[key] = minCost;

    return minCost;
}

int rand(int l , int h)
{
     return l + (h - l + 1) * (rand() / (RAND_MAX + 1.0));
}

#define N 30
Node* randomTree()
{
     int r = rand(1, N);
     int n = r;
     Node* root = new Node("root");
     Node* p = root;
     for (int i = 0; i < n; ++i) {
         r = rand(1, N);
         for (int j = 0; j < r; ++j) {
             std::ostringstream os;
             os << "c_" << i << '_' << j;
             Node* c = new Node(os.str());
             p->children.push_back(c);
             int s = rand(1, N);
             for (int k = 0; k < s; ++k) {
                std::ostringstream os2;
                os2 << os.str() << '_' << k;
                Node* v = new Node(os2.str());
                c->children.push_back(v);
             }
             p = c;
         }
     }

     return root;
}

int depth(Node* node)
{
    if (node->children.empty()) {
         return 1;
    }
    else {
         std::vector<int> cdepth;
         std::transform(
             node->children.begin(),
             node->children.end(),
```

```
            std::back_inserter(cdepth),
            depth
        );
        return 1 + *(std::max_element(cdepth.begin(), cdepth.end()));
    }
}

int size(Node* node)
{
    int s = 1;
    for (std::vector<Node*>::const_iterator
            it = node->children.begin(), itend = node->children.end();
            it != itend;
            ++it) {
        s += size(*it);
    }
    return s;
}

int size(Node* node, int level)
{
    if (level == 1)
        return 1;
    if (level == 2)
        return node->children.size();
    int s = 0;
    for (
         std::vector<Node*>::iterator
            it = node->children.begin(),
            itend = node->children.end();
         it != itend;
         ++it) {
         s += size(*it, level - 1);
    }
    return s;
}

void destroy(Node* n)
{
    if (!n->children.empty()) {
        std::for_each(n->children.begin(), n->children.end(), destroy);
    }
    delete n;
}

long bigO(Node* s, Node* t)
{
    int hmax = std::max(depth(s), depth(t));
    int o = 1;
    for (int i = 2; i <= hmax; ++i) {
        o += (1 + size(s, i)) * (1 + size(t, i));
    }

    return o;
}

int main()
{
    srand((unsigned)time(0));
    rand();
```

```
277     std::cout << "|s|,|t|,h(s),h(t),O(d),|d|,hits,#op,time" << std::endl;

279      for (int i = 0; i < 100; ++i) {
280          Node* s = randomTree();
281          Node* t = randomTree();

283          std::cout << size(s) << ',';
284          std::cout << size(t) << ',';
285          std::cout << depth(s) << ',';
286          std::cout << depth(t) << ',';
287          std::cout.flush();
288          std::cout << bigO(s, t) * 3 << ',';
289          std::cout.flush();
290          clock_t t1 = clock();
291          int delta = dist(*s, *t);
292          clock_t t2 = clock();
293          std::cout << dt.size() << ',' << hits << ',';
294          std::cout << delta << ',' << double(t2 - t1) / CLOCKS_PER_SEC;

296          std::cout << std::endl;
297          destroy(s);
298          destroy(t);
299          dt.clear();
300          ct.clear();
301          hits = 0;
302      }
303 }
```

Bibliography

[1] S. Abiteboul, J. McHugh, M. Rys, V. Vassalos, and J.L. Wiener. Incremental maintenance for materialized views over semistructured data. pages 38 – 49, New York, NY, USA, 1998. 14

[2] Mads Sig Ager, Olivier Danvy, and Henning Korsholm Rohde. Fast partial evaluation of pattern matching in strings. In *In Proceedings of PEPM'03*, pages 3–9, New York, NY, USA, 2003. ACM. 97

[3] AGG. The attributed graph grammar system. http://tfs.cs.tu-berlin.de/agg/. 14

[4] M Alanen and I Porres. Difference and union of models. *UML 2003 - The Unified Modeling Language, Proceedings*, 2863:2–17, 2003. 9, 11

[5] Carsten Amelunxen, Felix Klar, Alexander Königs, Tobias Rötschke, and Andy Schürr. Metamodel-based tool integration with MOFLON. In *30th International Conference on Software Engineering (ICSE'08)*, pages 807–810, Leipzig, Germany, May 2008. 11

[6] Michal Antkiewicz and Krzysztof Czarnecki. Design space of heterogeneous synchronization. In Ralf Lämmel and Joost Visser, editors, *GTTSE'07*, LNCS. Springer, 2008. 10, 11

[7] Apache. Axis 1.5 Web Services Engine. http://ws.apache.org/axis. 92

[8] ATL. *Specification of the ATL Virtual Machine version 0.1.* LINA and INRIA, Nantes, France, 2005. 11

[9] P. Tarr B. Hailpern. Model-driven development: The good, the bad and the ugly. *IBM Systems Journal*, 45(3):451–461, 2006. 1

[10] Wim Bast, Mariano Belaunde, Xavier Blanc, Keith Duddy, Catherine Griffin, Simon Helsen, Michael Lawley, Michael Murphree, Sreedhar Reddy, Shane Sendall, Jim Steel, Laurence Tratt, R. Venkatesh, and Didier Vojtisek. MOF QVT final adopted specification, Nov 2005. OMG document `ptc/05-11-01`. 11, 97, 106

[11] Philip Bille. A survey on tree edit distance and related problems. *Journal of Theoretical Computer Science*, 337:217–239, June 2005. 18

[12] Richard Bird and Oege de Moor. *Algebra of programming*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997. 136

[13] Xavier Blanc, Isabelle Mounier, Alix Mougenot, and Tom Mens. Detecting model inconsistency through operation-based model construction. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 511–520, New York, NY, USA, 2008. ACM. 9, 14

[14] Alex Borgida and Ronald J. Brachman. Conceptual modeling with description logics. pages 349–372, 2003. 13

[15] Antonio Cicchetti, Davide Di Ruscio, and Romina Eramo. Towards propagation of changes by model approximations. In *EDOCW '06: Proceedings of the 10th IEEE on International Enterprise Distributed Object Computing Conference Workshops*, page 24, Washington, DC, USA, 2006. IEEE Computer Society. 12

[16] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. Automating co-evolution in model-driven engineering. In *EDOC '08: Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference*, pages 222–231, Washington, DC, USA, 2008. IEEE Computer Society. 10

[17] K.T. Claypool and E.A. Rundensteiner. Sync your data: update propagation for heterogeneous protein databases. *VLDB Journal*, 14(3):300 – 17, Sept. 2005. 10, 14

[18] Charles Consel and Olivier Danvy. Tutorial notes on partial evaluation. In *POPL '93: Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 493–501, New York, NY, USA, 1993. ACM. 15

[19] Alexandre Correa and Claudia Werner. Applying refactoring techniques to UML/OCL models. In *Proc. Int'l Conf. UML*, volume 3273 of *LNCS*, pages 173 – 187. Springer-Verlag, 2004. 15

[20] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621 – 45, 2006/07/. 11, 41

[21] Krzysztof Czarnecki and Ulrich Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley Professional, Boston, MA, USA, June 2000. 3

[22] Gregory de Fombelle, Xavier Blanc, Laurent Rioux, and Marie-Pierre Gervais. Finding a path to model consistency. In *ECMDA-FA*, LNCS, pages 101–112, Bilbao, Spain, 2006. Springer Verlag. 13

[23] Wojciech J. Dzidek, Erik Arisholm, and Lionel C. Briand. A realistic empirical evaluation of the costs and benefits of UML in software maintenance. *Software Engineering, IEEE Transactions on*, 34(3):407–432, May-June 2008. 1

[24] Eclipse. Eclipse IDE. *http://www.eclipse.org*. 91

[25] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, March 2006. 14

[26] EMF. Eclipse Modeling Framework. http://www.eclipse.org/emf. 93

[27] M. Faid, R. Missaoui, and R. Godin. Knowledge discovery in complex objects. *Computational Intelligence*, 15(1), Jan 1999. 14

[28] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Trans. Program. Lang. Syst.*, 29(3):17, 2007. 9, 10, 11

[29] Yoshihiko Futamura. Parital evaluation of computation process - an approach to a compiler-compiler. *Systems, Computers and Controls*, 2(5):45–50, 1971. 155

[30] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, 1999. 14

[31] Holger Giese and Robert Wagner. Incremental model synchronization with triple graph grammars. In *Proc. of the 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS'06)*, pages 543–557. Springer Verlag, 2006. 10, 102

[32] Timothy Griffin and Leonid Libkin. Incremental maintenance of views with duplicates. pages 328 –, San Jose, CA, USA, 1995. 12

[33] J. Grundy, J. Hosking, and W.B. Mugridge. Inconsistency management for multiple-view software development environments. *IEEE Transactions on Software Engineering*, 24(11):960 – 81, 1998/11/. 12, 14

[34] Ashish Gupta, Inderpal Singh Mumick, and V.S. Subrahmanian. Maintaining views incrementally. volume 22, pages 157 – 166, Washington, DC, USA, 1993. 12, 14

[35] Jun Han. Supporting impact analysis and change propagation in software engineering environments. pages 172 – 182, London, UK, 1997. 14

[36] David Hearnden, Michael Lawley, and Kerry Raymond. Incremental model transformation for the evolution of model-driven systems. volume 4199 NCS, pages 321 – 335, Genova, Italy, 2006. 9, 11

[37] Markus Herrmannsdoerfer, Sebastian Benz, and Elmar Jürgens. Automatability of coupled evolution of metamodels and models in practice. In *MoDELS*, pages 645–659, 2008. 10

[38] I. Ivkovic and K. Kontogiannis. Using formal concept analysis to estalish model dependencies. In *Proceedings of the IEEE International Conference on Information Technology Coding and Computing*, Las Vegas, NV, Apr 2005. 1

[39] Igor Ivkovic and Kostas Kontogiannis. Towards automatic establishment of model dependencies using formal concept analysis. *International Journal of Software Engineering and Knowledge Engineering*, 16(4):499 – 522, 2006. 14

[40] S. Johann and A. Egyed. Instant and incremental transformation of models. *Automated Software Engineering, 2004. Proceedings. 19th International Conference on*, pages 362–365, Sept. 2004. 13

[41] Neil D. Jones. An introduction to partial evaluation. *ACM Comput. Surv.*, 28(3):480–503, 1996. 15, 97

[42] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial evaluation and automatic program generation*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993. xii, 15, 97, 99, 100

[43] Gerd Kamp, Holger Wache, and Consistency based Diagnosis. Using description logics for consistency-based diagnosis. Boston,MA, 1996. 13

[44] Anneke G. Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. 1

[45] Paul Klint, Ralf Lämmel, and Chris Verhoef. Toward an engineering discipline for grammarware. *ACM Trans. Softw. Eng. Methodol.*, 14(3):331–380, 2005. 10

[46] Julia L. Lawall. Faster fourier transforms via automatic program specialization. In *Partial Evaluation - Practice and Theory*, pages 338–355, London, UK, 1999. Springer-Verlag. 97

[47] Slaviša Marković and Thomas Baar. Refactoring ocl annotated UML class diagrams. *Software and Systems Modeling*, 7(1):25–47, February 2008. 15

[48] Tiago Massoni, Rohit Gheyi, and Paulo Borba. Formal refactoring for UML class diagrams. pages 152 – 168, 2005. 15

[49] Tiago Massoni, Rohit Gheyi, and Paulo Borba. Formal model-driven program refactoring. pages 362–376. 2008. 15

[50] Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Rondo: a programming platform for generic model management. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 193–204, New York, NY, USA, 2003. ACM. 9

[51] Tom Mens. On the use of graph transformations for model refactoring. volume 4143 NCS, pages 219 – 257, Braga, Portugal, 2006. 14

[52] Tom Mens, Gabriele Taentzer, and Olga Runge. Analysing refactoring dependencies using graph transformation. *Software and Systems Modeling*, 6(3):269 – 285, 2007. 14

[53] Tom Mens and Ragnhild Van Der Straeten. Incremental resolution of model inconsistencies. In *WADT 2006*, volume 4409 of *Lecture Notes in Computer Science*, pages 111–126. Springer-Verlag, 2007. 14, 15

[54] Tom Mens, Ragnhild Van Der Straeten, and Maja D'Hondt. Detecting and Resolving Model Inconsistencies Using Transformation Dependency Analysis. In *Proc. Int'l Conf. MoDELS 2006*, volume 4199 of *Lecture Notes in Computer Science*, pages 200–214. Springer-Verlag, October 2006. 14

[55] OMG. Model Driven Architecture (MDA) FAQ. http://www.omg.org/mda/faq_mda. 1

[56] OMG. "OMG Unified Modeling Language (OMG UML), Infrastructure, v2.1.2". http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF. 1

[57] Greg O Keefe. Dynamic logic semantics for UML consistency. In *ECMDA-FA*, LNCS, pages 113–127, Bilbao, Spain, 2006. Springer Verlag. 13

[58] Jung Gyu Park and Myong-Soon Park. Using indexed data structures for program specialization. In *In Proceedings of ASIA-PEPM '02*. 15

[59] Benjamin C. Pierce. *Types and programming languages*. MIT Press, Cambridge, MA, USA, 2002. 20, 174

[60] Ivan Porres. Rule-based update transformations and their application to model refactorings. *Software and Systems Modeling*, 4(4):368–385, Nov 2005. 9, 10

[61] Ali Razavi and Kostas Kontogiannis. Partial evaluation of model transformations. In *Proceedings of the 2012 34th ACM SIGSoft Internation Conference on Software Engineering*, ICSE '12. 6

[62] Ali Razavi and Kostas Kontogiannis. Prototalk: A generative software engineering framework for prototyping protocols in smalltalk. In *Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference - Volume 01*, COMPSAC '09, pages 435–442, Washington, DC, USA, 2009. IEEE Computer Society. 6

[63] Ali Razavi, Kostas Kontogiannis, Chris Brealey, and Leho Nigul. Incremental model synchronization in model driven development environments. In *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '09, pages 216–230, New York, NY, USA, 2009. ACM. 6

[64] S.P. Reiss. Incremental maintenance of software artifacts. *IEEE Transactions on Software Engineering*, 32(9):682 – 97, Sept. 2006. 12, 14

[65] John C. Reynolds. *Theories of programming languages*. Cambridge University Press, New York, NY, USA, 1999. 174

[66] Mehrdad Sabetzadeh and Steve M. Easterbrook. Analysis of inconsistency in graph-based viewpoints: A category-theoretic approach. In *18th IEEE International Conference on Automated Software Engineering (ASE 2003)*, pages 12–21, Montreal, Canada, October 2003. IEEE Computer Society. 14

[67] Andy Schürr. Specification of graph translators with triple graph grammars. volume 903 of *LNCS*, pages 151–163, Herrsching, Germany, June 1994. 11

[68] Bran Selic. The pragmatics of model-driven development. *IEEE Software*, 20(5):19–25, 2003. 1

[69] Peter H. Sellers. An algorithm for the distance between two finite sequences. *Journal of Combinatorial Theory, Series A*, 16(2):253–258, 1974. 28

[70] Mary Shaw. "self-healing": softening precision to avoid brittleness: position paper for WOSS '02: workshop on self-healing systems. In *WOSS '02: Proceedings of the first workshop on Self-healing systems*, pages 111–114, New York, NY, USA, 2002. ACM. 2

[71] G. Sittampalam, O. de Moor, and K.F. Larsen. Incremental execution of transformation specifications. volume 39, pages 26 – 38, Venice, Italy, 2004/01/. 12

[72] Ragnhild Straeten, Viviane Jonckers, and Tom Mens. A formal approach to model refactoring and model refinement. *Software and Systems Modeling*, 6(2):139 – 162, 2007. 13

[73] Ragnhild V. Straeten, Tom Mens, Jocelyn Simmonds, and Viviane Jonckers. *Using Description Logic to Maintain Consistency between UML Models*, pages 326–340. 2003. 13, 15

[74] R. S. Sundaresh. Building incremental programs using partial evaluation. In *In Proceedings of PEPM'91*, pages 83–93, New York, NY, USA, 1991. ACM. 15

[75] Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of ACM*, 26:422–433, July 1979. 27

[76] Laurence Tratt. Model transformations and tool integration. *Journal of Software and Systems Modeling*, 4(2):112–122, May 2005. 12

[77] Ragnhild Van Der Straeten, Viviane Jonckers, and Tom Mens. Supporting model refactorings through behaviour inheritance consistencies. In Ana Moreira Thomas Baar, Alfred Strohmeier, editor, *UML 2004 - The Unified Modeling Language*, volume 3273 of *Lecture Notes in Computer Science*, pages 305–319. Springer-Verlag, October 2004. 15

[78] Guido Wachsmuth. Metamodel adaptation and model co-adaptation. In Erik Ernst, editor, *Proceedings of the 21st European Conference on Object-Oriented Programming (ECOOP'07)*, volume 4609 of *Lecture Notes in Computer Science*, pages 600–624. Springer-Verlag, July 2007. 10

[79] Glynn Winskel. *The formal semantics of programming languages: an introduction*. MIT Press, Cambridge, MA, USA, 1993. 174

[80] WTP. Eclipse Web Tools Platform. http://www.eclipse.org/webtools. 91

[81] Yingfei Xiong, Dongxi Liu, Zhenjiang Hu, Haiyan Zhao, Masato Takeichi, and Hong Mei. Towards automatic model synchronization from model transformations. In *ASE'07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 164–173, New York, NY, USA, 2007. ACM. 9, 10, 11

[82] Yingfei Xiong, Haiyan Zhao, Zhenjiang Hu, Masato Takeichi, Hui Song, and Hong Mei. Beanbag: Operation-based synchronization with intra-relations. Technical Report

GRACE-TR-2008-04, Center for Global Research in Advanced Software Science and Engineering, December 2008. 9, 11